User Guide

Spirent iTest ®

Version 7.2 Rev A

October 2018

DOC10692

Spirent Communications, Inc.
27349 Agoura Road
Calabasas, CA
91301 USA

Copyright

Limited Warranty

Spirent Communications, Inc. (“Spirent”) warrants that its Products will conform to the description on the face of order, that it will convey good title thereto, and that the Product will be delivered free from any lawful security interest or other lien or encumbrance.

Spirent further warrants to Customer that hardware which it supplies and the tangible media on which it supplies software will be free from significant defects in materials and workmanship for a period of twelve (12) months, except as otherwise noted, from the date of delivery (the “Hardware Warranty Period”), under normal use and conditions.

To the extent the Product is or contains software (“Software”), Spirent also warrants that, if properly used by Customer in accordance with the Software License Agreement, the Software which it supplies will operate in material conformity with the specifications supplied by Spirent for such Software for a period of ninety (90) days from the date of delivery (the “Software Warranty Period”). The “Product Warranty Period” shall mean the Hardware Warranty Period or the Software Warranty Period, as applicable. Spirent does not warrant that the functions contained in the Software will meet a specific requirement or that the operation will be uninterrupted or error free. Spirent shall have no warranty obligations whatsoever with respect to any Software which has been modified in any manner by Customer or any third party.

Defective Products and Software under warranty shall be, at Spirent's discretion, repaired or replaced or a credit issued to Customer's account for an amount equal to the price paid for such Product provided that: (a) such Product is returned to Spirent after first obtaining a return authorization number and shipping instructions, freight prepaid, to Spirent's location in the United States; (b) Customer provides a written explanation of the defect or Software failure claimed by Customer; and (c) the claimed defect actually exists and was not caused by neglect, accident, misuse, improper installation, improper repair, fire, flood, lightning, power surges, earthquake, or alteration. Spirent will ship repaired Products to Customer, freight prepaid, based on reasonable best efforts after the receipt of defective Products. Except as otherwise stated, any claim on account of defective materials or for any other cause whatsoever will conclusively be deemed waived by Customer unless written notice thereof is given to Spirent within the Warranty Period. Spirent reserves the right to change the warranty and service policy set forth above at any time, after reasonable notice and without liability to Customer.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, ARE HEREBY EXCLUDED, AND THE LIABILITY OF SPIRENT, IF ANY, FOR DAMAGE RELATING TO ANY ALLEGEDLY DEFECTIVE PRODUCT SHALL BE LIMITED TO THE ACTUAL PRICE PAID BY THE CUSTOMER FOR SUCH PRODUCT. THE PROVISIONS SET FORTH ABOVE STATE SPIRENT'S ENTIRE RESPONSIBILITY AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY BREACH OF ANY WARRANTY.

Contents

Welcome to iTest

iTest Overview

Welcome to fast, easy device testing

iTest is the first testing tool to help device testers do their jobs faster and more easily than ever before. Until now, device testers spent hours each day performing manual and repetitive tasks — from setting up devices and configuring test beds, to communicating test reports. They had no choice but to rely on manual tests and scripting to test their devices. iTest is the only solution that solves this problem by simplifying the testing process. With iTest, device testers gain more time to test and ensure product quality.

iTest does not change the way you test. Rather, it captures and records all testing information — from device setup to results — so testers and developers of all skill levels can repeat tests quickly and accurately. iTest also generates detailed, portable reports that clearly document test actions and corresponding responses, so anyone can reproduce the bug — and validate that it was fixed. With iTest, you can rapidly test device features across multiple protocols and platforms, and keep pace with development.

- iTest is a powerful testing tool for a manual or feature tester or anyone developing tests at the beginning of the automation “food chain”.
- iTest is an integrated test environment for management interfaces like CLI (Telnet or SSH), Web, SNMP, Tcl or Python API. While testing with these interfaces, iTest captures every command and response, effectively logging all activity. Users can simply click a button to replay any combination of captured steps across any set of management protocols.
- For commonly performed tasks like reconfiguring a testbed, steps can be saved as a “Favorite” procedure and replayed at any time. By combining these Favorites, users can perform complex tasks with a single mouse-click, enabling a significant improvement in productivity.
- Since iTest is always logging all test activity, defects are captured as they happen and can easily be saved as a detailed report of an observed bug. No need to go back and try to duplicate a bug – just hit Replay to verify that it is reproducible.
- For users doing unit testing or developing smoke tests, procedures can be saved and easily replayed to verify device functionality. Procedure editing allows for modifications without having to re-capture.
- If test steps need to be documented, iTest automatically captures detailed commands and responses for multiple protocols – allowing users to save these as the basis for a future, fully-automated test case.

  Spirent provides Python Automation Library that may be integrated at a step level into your python-based automation scripts and suites. In addition, iTest uses the Python Automation Libray to export session capture steps as Python scripts so that you can run these steps inside a larger Python test suite. iTest builds Python syntax to assist you in quickly authoring your test cases using the Spirent Python Automation Library.

Be sure to view the *Getting Started* video tutorials to learn the iTest basics. This is by far the best and fastest way to learn how to use iTest.

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Watch this video to learn about Session Builder to build custom sessions: https://www.youtube.com/watch?v=80H6f5bn-Vo

Session types

iTest supports the following session types

Built-in Session Types

Chat (XMPP Chat)	You can add steps that receive and send XMPP chat messages during execution. A test case can send and receive as many messages as are needed. Message text can contain both fixed text and response data
Command Prompt	You can run Microsoft Windows Command Prompt sessions (that is, run cmd terminal sessions). For each open session, the Command Prompt Session window displays your commands and the local PC's responses.
Database Client	The Database Client session window is an interactive browser where you perform database operations and monitor responses. iTest captures all commands and responses and you an save captured items as test case steps that start the session and set and request database records. Automated test cases open the same session window to perform database operations. iTest supports sessions with MySQL, SqlServer, Oracle, Szlite, Derby, or a custom database type that you specify.
File	The File session type enables an automated test case to work with text files during execution (open a file, go to a specified position in the file, read a line, write a line, and so on).
HTTP	Using an HTTP session, a test case can talk directly with a device using the HTTP protocol operations **GET** and **POST**. HTTP **GET** commands are useful in cases where you are not testing a Web application, but rather are testing something like a device through which the HTTP is passing.
SSH	A virtual terminal that communicates with a device using the SSH protocol (SSH-1 or SSH-2) defined in RFC 4250 **Note** Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet or SSH session to “localhost” to access a shell.
Swing	The Swing session type allows you to test Java applications with user interfaces that were developed using Swing.
Tcl Shell	You can type Tcl commands and expressions into the iTest Tcl Shell session window. iTest captures your commands and the interpreter's responses.

Telnet	A virtual terminal that communicates with a device using the Telnet protocol defined in RFC 854 **Note** Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet or SSH session to “localhost” to access a shell.
Python	iTest Python session is a terminal session, similar to Tcl Shell. The session uses native Python interpreter to getting responses. Supported versions of interpreters are: 2.4-2.7 and 3.0-3.6. iTest support an internal Python interpreter and also allows you to point to an external interpreter via Preferences settings (see “Setting preferences for Python” on page 1166).

Test Equipment Integration

iTest supports a broad variety of traffic and test equipment by several suppliers.

Each traffic generator session window is an interactive terminal where you enter commands to the device and the device returns text responses. Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from traffic generator devices.

Ixia IxLoad	To run an IxLoad session in iTest, you first define a test or tests in IxLoad and save the configuration file (RFX file) as you normally would. You then start the IxLoad session in iTest and run the IxLoad test. When the session starts, it checks out the libraries and connects and runs the initialization script and then loads the configuration, validates it, and transforms it (see the description for the iTest **load** command). The test produces responses and CSV files, which you can then post-process. The IxLoad session window is an interactive terminal where you enter commands to perform IxLoad actions on the device. IxLoad returns text responses. The responses to IxLoad responses are structured and iTest uses built-in mappers to supply read-made queries.
IxNetwork	In IxNetwork sessions, you can start and stop traffic, start and stop capture, and review statistics. The levels of the Object Data Matrix are represented as subdirectories (see Table 1-11, API Command Data Model Structure in the IxNetwork Tcl API Guide). The iTest IxNetwork session window enables you to navigate the object hierarchy by using commands that you typically use to navigate a file system.
Ixia N2X	The Ixia N2X session window is an interactive terminal where you enter commands to perform Ixia N2X actions on the device. Ixia N2X returns text responses. Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Ixia devices. iTest captures both the text response and a structured version of the response. iTest auto-generates appropriate queries for response data, so you can easily work with or analyze data of interest in the response
Ixia Traffic	The Ixia Traffic session window is an interactive terminal where you enter commands to perform Ixia Traffic actions on the device. The session on the Ixia device returns text responses.
Spirent Avalanche	There are several options for running an Avalanche test in iTest: • Running a test using the Tcl scripts generated by Avalanche • Running an Avalanche test on a TestCenter device • Running a test using Demo mode

Spirent Landslide NTAF	You can use iTest to automate Spirent Landslide tests. In iTest, when you start a Landslide session, iTest launches the Landslide TAS user interface running on the Landslide device. 1. Now you can interact with the TAS in the normal way. For example, you might load a test configuration, start the test session, collect the responses, wait to collect several data sets, stop the test session, request the test session results, and then close the test session. 2. When you are finished working on the TAS, you return to iTest and close the Landslide session window. 3. You can now save the captured steps and responses as an iTest test case. As needed, you can modify and update the test case (for example, modify the **ImportTestSuite** step by causing it to load a different Landslide test suite file or replace the filename with a variable whose value is set at runtime).
Spirent Landslide REST	iTest integrates with Spirent Landslide and provides REST API to ensure that the automation functionality is readily available to a wide variety of clients using both script and GUI. This integration also eliminates the requirement of a Landslide installation on the client system. This allows you to use existing tools available to create Landslide automation clients that can run on any platform. • The iTest REST API session communicates with Landslide TestServer via Landslide Lab server using the REST API (Landslide Lab is required when working with the session). • The iTest REST session can run on multiple platform without requiring the Landslide libraries. • Since all actions and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent Landslide. Any action that you perform in the iTest Landslide session is forwarded to Spirent Landslide running on the device. Landslide performs the action and returns its normal response. You can view the response in the Results section on the iTest window, just like you do in Landslide.
Spirent SmartBits	The SmartBits session window is an interactive terminal where you enter commands to perform SmartBits actions on the device. SmartBits returns text responses.
Spirent TestCenter GUI	This session type appears in iTest only to support old test cases.
Spirent TestCenter REST	iTest integrates with Spirent TestCenter (STC) and provides REST API to ensure that the automation functionality is readily available to a wide variety of clients using both script and GUI. This integration also eliminates the requirement of an STC installation on the client system. This allows you to use existing tools available to create STC automation clients that can run on any platform.

Other Session Types

ADB (Android Debug Bridge) (Custom session)	The ADB (Android Debug Bridge) custom session allows you to communicate with an emulator instance or connected Android-powered device.
CloudStress (Custom session)	The iTest CloudStress session provides way to automate testing and performance verification of your cloud infrastructure, without having to know the scripting languages.
EggPlant	EggPlant Test session provides ways of automating execution of image-based GUI testing. The seamless testing of these iTest functionalities is achieved by iTest integration with EggPlant application (https://eggplant.io/eggplant-integrations/).

Flex/Flash	iTest supports testing Flash applications that were developed using Adobe Flex. iTest can capture your interactions with Flex applications that are hosted on web pages. iTest captures each step of an interactive (manual) test in the Flex application.
REST	The REST session window provides a work surface for composing and submitting HTTP requests. Any request that you submit in the iTest session is forwarded to the RESTful service. The service performs the action and returns its normal response. iTest captures all of the actions that you perform in a session and all of the responses.
RFT — Rational Functional Tester	The session window for iTest RFT sessions has been designed in partnership with IBM to enable you to launch Rational Functional Tester scripts from iTest. This integration is certified by IBM as *Ready for IBM Rational Software*.
RQM — Rational Quality Manager	You can create a Rational Quality Manager (RQM) test case that executes a iTest test case. When you choose to run an automated test from within RQM, RQM sends commands to the devices in the test lab, causing the iTest test case to run and the results to be displayed in your browser.
SNMP	A hierarchical browser for getting and setting SNMP MIB data using the Simple Network Management Protocol (SNMP V1, V2c, or V3) defined in RFC 1157
Web Services	The Web Services session window provides a work surface for composing and submitting Web Service requests. Any request that you submit in the iTest session is forwarded to the Web Services server. The Web Service performs the action and returns its normal response.
Mail (SMTP)	You can add steps that construct and send email messages during execution. A test case can construct and send as many email messages as are needed. The message body can contain both fixed text and test response and result data.
Mail (POP3)	You may se the Mail (POP3) sessions to retrieve emails from subscribers, view content, extract the required text and attachments, save the attached images as individual files, and then insert them into other sessions, for example, Selenium.
NetConf	The NetConf session window displays your commands and the device's responses. You can think of the session window as a terminal — a terminal to a NetConf service as a subsystem that iTest is monitoring and capturing.
OpenStack Neutron (Custom sessions)	The OpenStack Neutron session allows you to manage OpenStack Neutron Network such as router, network, subnet, port, security group, virtual network topologies including services such as firewalls, load balancers, and virtual private networks (VPNs), via RESTfull HTTP service.
Process	Execute and manage processes on the computer that is running iTest.
PowerShell	PowerShell sessions execute commands at the Windows PowerShell prompt. The PowerShell session window displays your commands and the local PC's responses.
Ranorex	Ranorex Test session provides ways of automating the Windows, Web, and Mobile UI applications. The seamless testing of these iTest functionalities is achieved by iTest integration with Ranorex application (www.ranorex.com).
Serial Port (mandatory)	For Serial Port sessions, the computer running iTest communicates directly over a serial port connection with the device under test. For each open session, the Serial Port session window displays your commands and the device's responses. You can think of the session window as a terminal client — a terminal that iTest is monitoring and capturing.
Selenium	For Selenium sessions, iTest opens an instance of the Firefox browser. You interact with the pages in the normal way while iTest captures your actions and responses from the session.

Syslog	Each Syslog session monitors the syslog messages that arrive at the built-in iTest syslog server (visible in the Syslog view). While the syslog server receives all messages, any syslog session can filter the messages based on the following property settings in the session profile. As a result of configuring session profile settings, only the messages that meet the filter settings appear in the session window. This enables your test cases to analyze the particular responses (messages) of interest and to ignore irrelevant messages.
Wireshark	Wireshark sessions provide a command line interface for interactively capturing packets from a network interface. For commands that return status and packet data, iTest saves the responses as structured data and generates associated queries to simplify pass/fail analysis.
TL1	There are two types of TL1 interface: • Automation interface — These interfaces are not meant for human interaction. These interfaces do not return a prompt and might not even echo what the user types. • Hybrid interface — These interfaces echo what the user types and return a prompt. The prompt can be a normal one like login: or mydut> or could be TL1 end of message: < or ;
UDP	For UDP sessions, the computer running iTest communicates directly over UDP (User Datagram Protocol) with the specified device. For each open session, the UDP session window displays your commands and the local echo. Open another session to view the device's response. You can think of the window as a terminal client — a terminal that iTest is monitoring and capturing.
VNC	iTest VNC sessions are intended to help you to control a remote OS to perform configuration/setup/tear-down tasks. iTest VNC sessions are not intended to enable you to thoroughly test an application on a remote platform.
VMware vSphere Client	The session window for vSphere sessions in iTest has been designed in partnership with VMware to closely resemble the vSphere client. As a result, you can capture vSphere steps using iTest without having to learn a new interface or new command names. You interact with the iTest session almost exactly like you interact with vSphere. The iTest interface to vSphere enables you to perform and automate a wide range of tasks
XML-RPC	The XML-RPC session window provides a work surface for composing and submitting XML-RPC method calls over HTTP and HTTPS. Any request that you submit in the iTest session is forwarded to the XML-RPC service. The service performs the action and returns a response. You can view the response in the Response section of the XML-RPC session window. iTest captures all of the actions that you perform in a session and all of the responses. You can use the captured items to create test case steps that interact with the XML-RPC server.

Other integration

Zephyr for JIRA	**Zephyr for JIRA** Test Management system with the add-on **ZAPI**, integrated with **iTest** makes it easier to share projects across the organization. The integration (**Zephyr for JIRA** and **iTest**) allows you to work with JIRA tests via RESTful APIs to perform these tasks.
EGit	iTest integrates Eclipse EGit plugin, which allows you to use Git source control from iTest. The instructions provided in this guide assumes the following: • Groups within your organization want to work on a project using EGit to develop and maintain test cases in a single location (the **master branch**). • Users will have their own local repository (a copy of the test cases code including all the source control relevant information). • Each user will receive changes and send changes using a remote repository at GitHub. For details about EGit see http://wiki.eclipse.org/EGit/User_Guide.

iTest licensing

Overview

This chapter explains how users of iTest can configure and view licensing settings. To set up licensing so you can use iTest, you perform two simple tasks:

- Specify the host computer that runs the license server software
- Specify which edition of iTest and its modules to use

Note License server software provides licenses for users of iTest. Information on configuring the server appears in the *iTest Installation Guide* (a PDF document that appears in the Spirent Knowledge Base).

To configure licensing settings

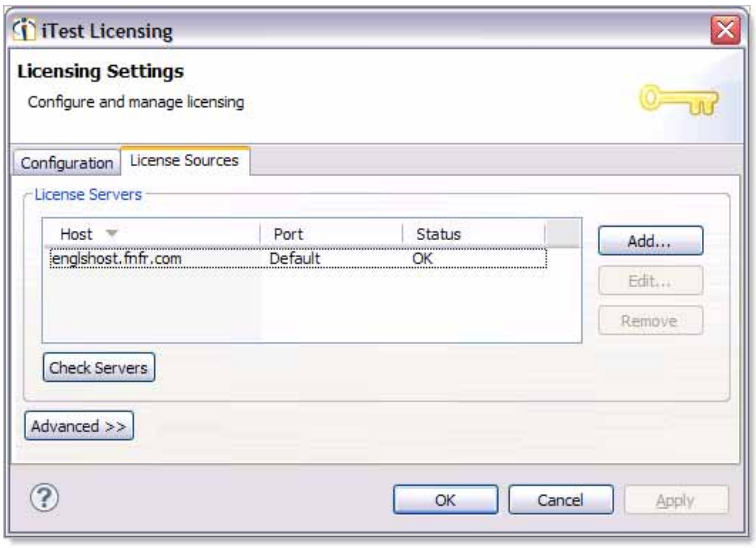
While running iTest, click **Help > Configure iTest Licensing** and then perform the following steps:

Important Start iTest while logged in using an account with your normal working privileges. As with any application, if you create a document having logged in using an account with **Administrator** privileges and then later try to use the document while logged in using an account with different privileges, iTest will not allow access to the document.

Step 1 Specify the license server host computer

Typically, your organization has purchased several iTest licenses that "float" to users as needed. When you start iTest, the license server software (managed by your iTest administrator) allocates one floating license for each edition and module selected for your computer to use for the duration of the session.

1 Click the **License Sources** tab.



2 In the **License Servers** section, click **Add**.

3 Specify the DNS hostname or IP address of the license server host (provided by your administrator). Optionally, specify the **Port**.

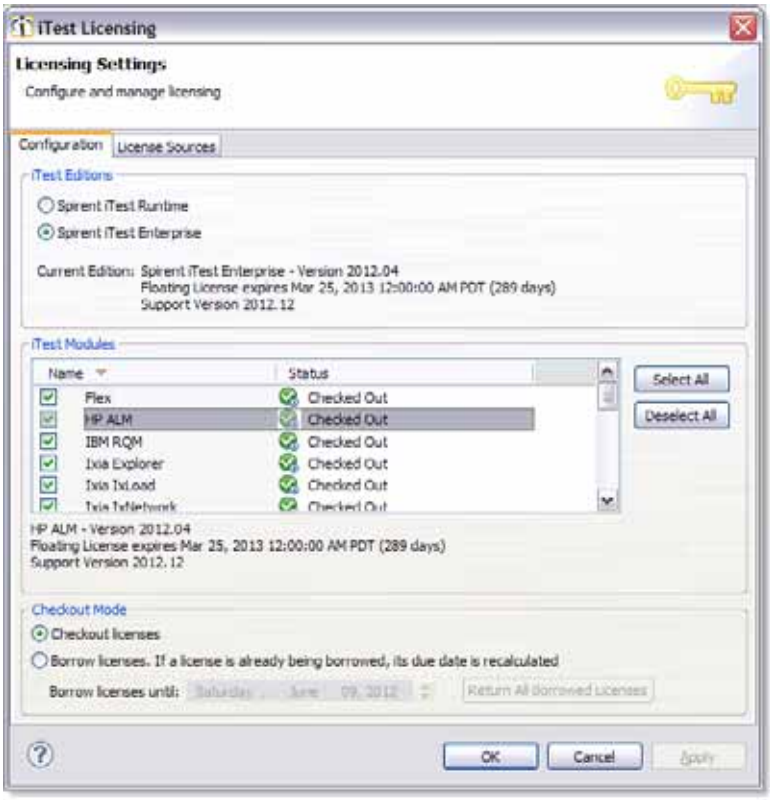
4 Click **OK**.

As needed, click **Check Servers** to view the status of license servers.

Note You can change the settings at any time.

The **Advanced** section is used only by Spirent Customer Support.

Step 2 Specify which edition of iTest and its modules to use



1 On the **Configuration** tab, select the edition of iTest to use. The page displays the edition of the iTest software, the version (based on the release date), and the end dates of your software license and Spirent Customer Support agreement.

2 To enable a feature, check one or more modules (typically, you check all modules).

To view the version of the module and the end dates of your license and support agreement, select the module in the list (we selected **HP ALM** in the example).

Note You have the option to borrow a license for cases when you cannot connect to the license server (for example, at home on the weekends or running tests at a vendor's or customer's site). See "Borrowing a license" on page 11.

Borrowing a license

In some common situations (at home on the weekends, running tests at a vendor's or customer's site) you cannot obtain a iTest license because you cannot connect to the license server. In these situations, you can still use iTest by "borrowing" a license from the license server onto a portable computer for a limited time and then "returning" it when you no longer need it.

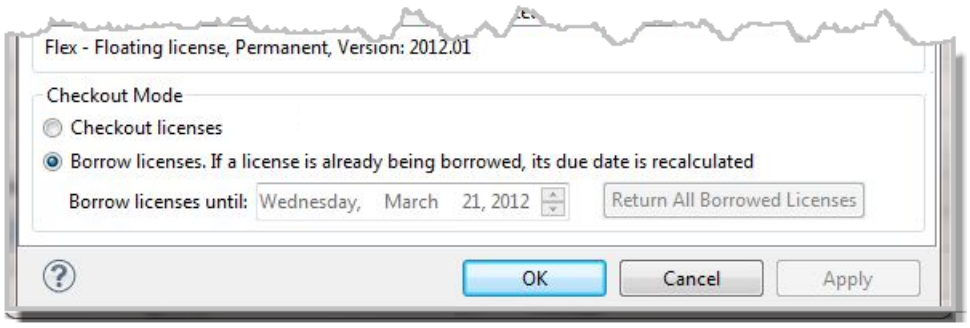
To borrow a license

1 Temporarily connect the portable computer that will use the borrowed licenses to the license server.

2 Start iTest and open the **iTest Licensing Settings** dialog box by clicking **Help > Configure iTest Licensing**.

3 In the **iTest Modules** section, check the licenses that you want to borrow (click **Select All** or **Unselect All** to speed the process).

4 In the **Checkout Mode** section, check **Borrow Licenses** and specify the "borrow until" date — the last day that you expect to use the borrowed license. You can either type the date directly or select the month, day, or year and use the up/down controls.



Note Some feature licenses may not be "borrowable" or may have limits on how long they can be borrowed. The default time limit is one week unless altered by the license terms. If a license is already being borrowed on the current computer, its "borrow until" date is recalculated.

5 Click **OK**. The licenses are now borrowed onto the computer.

The icon in the **Status** column changes to reflect borrowing and the "borrow until" date is displayed. iTest will now operate on the portable computer (not connected to the license server) using the borrowed license until the expiration date.

To return a borrowed license

1 Re-connect the computer that is borrowing licenses to the license server.

2 Start iTest and open the **iTest Licensing Settings** dialog box by clicking **Help > Configure iTest Licensing**.

3 In the **Checkout Mode** section, click **Return Borrowed Licenses**. iTest returns all selected licenses to the license server and then immediately checks them out.

Note If iTest has no connection to the license server when you click **Return Borrowed Licenses,** iTest will not disable licenses locally. Instead, displays a message saying that the license server does not respond, check the connection and try again.

You now have the following options:

- Leave the computer connected to continue to use iTest with licenses that are checked out from the license server (the "normal" mode)
- Exit iTest to release all of the iTest licenses to the license server for use by other computers and users.

CHAPTER 2

Activity Pages

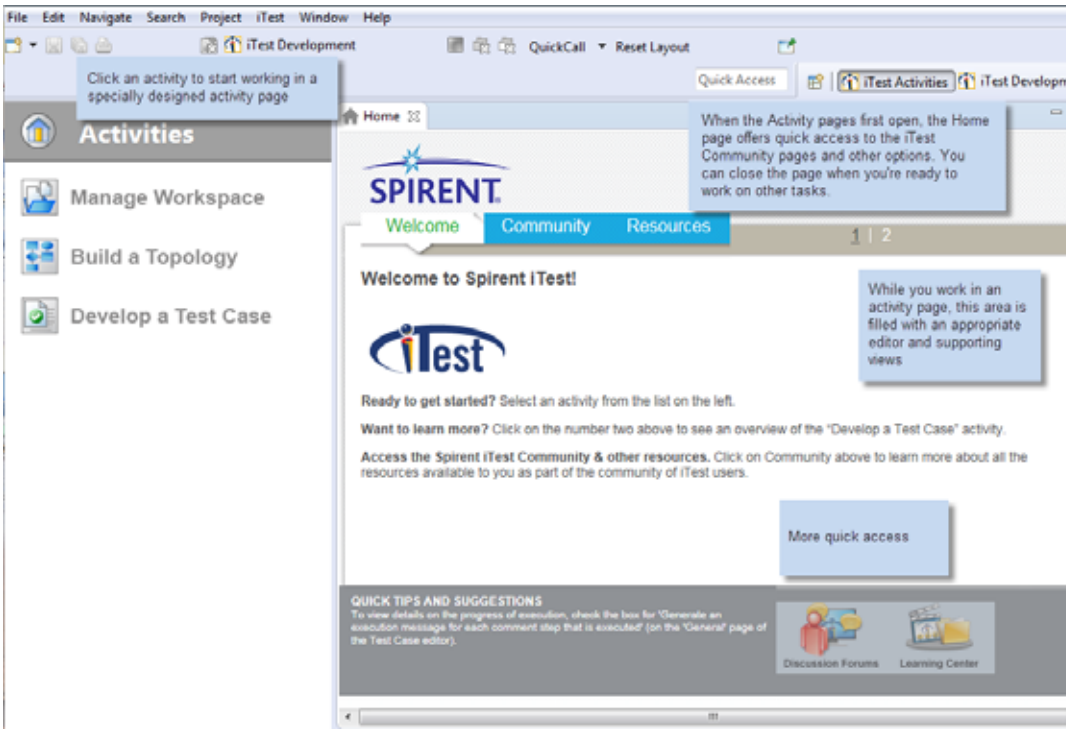
Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Activity pages

Because you can use iTest for a large variety of tasks (for example, performing manual testing on a CLI terminal or Web or SNMP browser, capturing the test sessions, developing and debugging automated test cases, specifying the portions of response data to validate, executing test cases, monitoring execution, and reviewing test reports), the iTest window can become pretty complicated.

Let's take a look at the perspective that makes it easier to perform your most common activities, the **iTest Activities** perspective (it opens the very first time that you start iTest). Your most common activities are available with one click on the **Activities** tab.



In this chapter

Managing your workspace

Managing your workspace

To facilitate use of approved shared files, your organization might ask you to import one or more existing projects or to create a new project or create a reference session profile. You perform these activities only occasionally on an as-needed bases using the **Manage workspace** page. See "Importing a project" on page 35, "Creating Reference Session" on page 36, and "Creating a project" on page 37 for details.

Setting preferences for the Activity pages

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Activities**.

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Preferences: Spirent > Activities

Close editors when exiting an activity	When you navigate away from an activity page (like **Build a Topology**), iTest closes all active editors by default. Default: checked
Close sessions when exiting an activity	When you navigate away from an activity page (like **Build a Topology**), iTest exits all sessions and closes all active session windows by default. Default: checked
Do not prompt me about closing sessions when exiting an activity	When you navigate away from an activity page, a dialog box notifies you that iTest will exit all sessions and close all active session windows by default. Check the box to not display the dialog box Default: unchecked
Close views when exiting an activity	When you navigate away from an activity page (like **Build a Topology**), iTest closes all views by default. Default: checked
Allow iTest Home Page to use dynamic web-based content	By default, the **Home** page loads from the iTest Web site. Uncheck the box to load a web page that is stored in the iTest installation directory on your local hard disk. Default: checked

Building a topology

Building a topology

The **Build a topology** page provides quick access to the tools that help you to create and edit topologies (as described in Chapter 24, “iTest Topology editor”):

- Working on the topology in the **Topology** editor
- Adding a session configuration to a device in the topology or to a testbed
- Starting (launching) a session with a device in the topology

Messages

The **Messages** table lists all issues with devices and sessions in the topology. Double-click a message to select the device that has the issue. You can then use the tools in the Topology editor to fix the issue.

1 When you click **Finish**, the **Build a Topology** activity page opens.

- **To create a new topology or open an existing one**

  See “Creating a new topology or opening an existing one” on page 16.

- **To start (launch) a session with a device in the topology**

  1 On the **Build a topology** activity page, click **Start Session**.

2 The **Start Session** dialog box opens. In the tree, open the device, select the session configuration, and then click **Start**. The session starts in the editor area.

Tip While locating the session, to limit the number of devices that appear in the tree, type filter text in the box. Only session configurations that include the filter text are displayed.

- **To add a session configuration for a device**

  See "Adding a session configuration for a device" on page 16.

Creating a new topology or opening an existing one

When you select the **Build a topology** activity page, you first specify whether to work on an existing topology or to create a new one.

- **To work on an existing topology or create a new one**

  On the **Open Topology** wizard, select either **Select an existing topology** or **Create a new topology**. The dialog box changes based on your selection.

  - To work on an existing topology, navigate to the document and click **Finish**.
  - To create a new topology, specify the following settings and click **Finish**.

Save in	Navigate to the project and folder to create the topology in. Typically, you create a topology in the default project named **my_project** in the default folder named **topologies**.
File name	Type a name for the new document. Topology files use the **.tbml** filename extension.

Adding a session configuration for a device

1 On the Topology editor, select the device.

2 On the **Build a Topology** page, click **Add Session**.

3 On the **Add Session** dialog box, iTest provides default values for the following properties; you can modify the values as needed

Device	This is the name of the device that you had selected. By default, the session is defined for the device that you selected on the canvas of the Topology editor. All sessions defined for devices in the topology appear in the drop-down list. If you select a different device, the session configuration is added to the device.
Profile name	Provide a **name** for the session configuration, consider using a filename convention like **<session type>_<Session name>.**

Editing a Session Configuration for a device

Important Session profiles that you define for devices in a topology are saved as part of the topology file and are not saved as separate session profile documents. To edit a session configuration for a device, see "Add, edit, or remove a session configuration for a topology device" on page 500.

Developing a test case

The **Develop a test case** page provides quick access to the following operations:

- Create a new test case or open an existing test case
- Adding steps by capturing (start sessions with devices and save the captured steps into the test case)
- Execute the test case
- Add analysis rules that validate a value in a response or store it into a variable
- View test reports for the current test case

To create a new test case or open an existing test case

When you select the **Develop a test case** activity page, you first specify whether to work on an existing test case or to create a new test case.

1 On the **Develop a test case** dialog box, select either **Select an existing test case** or **Create a new test case**. The dialog box changes based on your selection.

- To work on an existing test case, navigate to the document and click **OK**.
- To create a new test case, specify the following settings and click **OK**.

Save in	Navigate to the project and folder to create the test case in. Typically, you create a test case in the default project named **my_project** in the default folder named **test_cases**.
File name	Type a name for the new document. Test case files use the **.fftc** filename extension.
Owner	Optional. Type the unique identifier of the person responsible for developing and/or maintaining the test case (typically, the name, login name, or email address), Default: current username
Headline	Optional. Type a one-line description that documents the usage and function of the test case. In addition to the locations mentioned earlier, this text also appears in the Favorites view to help you when selecting a test case.
Description	Optional. Type additional text that describes the test case to make its usage clear to coworkers. **Tip** This is an excellent place to paste a copy of the test plan.

Use test case template	Check the box to add the steps and properties of the template test case into the new test case. For information on creating a template test case, see "Creating a template for new test cases" on page 142. Default: unchecked
Associate a topology with the test case	Optional. Specify a topology or testbed to use for the sessions in the test case. When you specify a topology, you can make the test case more flexible by using parameters defined in the associated session profiles to customize behavior at runtime. If the **Associate a topology with the test cased** property is blank or no Global topology or Global testbed is specified, then **device** URIs are not supported in **open** steps. **Test documentation includes the topology or testbed** When a topology or testbed is used for execution, to help you keep track of which topology or testbed was used, an informational execution message will appear immediately after the execution start message. The message identifies the fully qualified URI of the topology or testbed that is being used. In addition, the URI appears in the header section of the test report.

2 When you click **OK**, the **Develop a test case** activity page opens.

To add steps by capturing (start sessions with devices and save the captured steps into a test case)

1 After opening or creating the test case, on the **Develop a test case** activity page, click **Add steps by capturing**. The **Add steps to Test Case** page opens.

2 The current test case is identified at the top of the page. If a topology or testbed file or parameters file is associated with the test case, then they are also identified.

Click the filename to open the file in the appropriate editor

3 Now start a session with a device: Click **Start a Session** . The **Start a Session** dialog box displays all devices defined in the topology or testbed that you associated with the test case. Navigate the tree to select the appropriate session and then click **Start**.

4 Using the session profile settings that are defined for the device, iTest launches a session on the device.

- You can open as many sessions on as many devices as you like.
- Captured sessions are listed in the **Sessions** table.

- Captured steps are listed in the **Steps** table.

iTest captures any commands that you perform whenever the capture indicator displays **Capturing**
Captured steps are listed in the **Steps** table.

To pause capture, click **Pause** . The capture indicator changes to **Paused** .
While capture is paused, any commands that you submit in any session are not captured.

To restart capture, click the **Start/Restart Capture** button. iTest resumes adding steps to the **Steps** table.

5 Here are some tools for documenting or clearing captured steps

Insert Comment	Select a step and click the button to add a comment step after the selected step. The comment step is added to the test case along with the captured steps.
Clear Steps	Clears all captured steps from the table, closes all open sessions, and cancels the capture activity.

6 When you are ready to add the captured steps from the **Steps** list into the test case, click **Add Steps** .

- To ***not*** add a particular step or session into the test case, right-click the step and select **Exclude Item**. Excluded steps are not added to the test case.

  By default, excluded steps are not displayed in the table. To display excluded steps in the list (highlighted to indicate that they will not be added to the test case), right-click in the table and select **Show Excluded Items**.

- For new test cases, steps are added to the end of the test case.
- For existing test cases, the **Insert Steps** wizard prompts you to specify the location in the test case to add the steps.
  - By default, the **Close open sessions** check box is checked; you can uncheck it to allow the sessions to remain active after you complete the process of adding the steps into the test case.
  - When you click **Finish**, the steps are added to the test case.

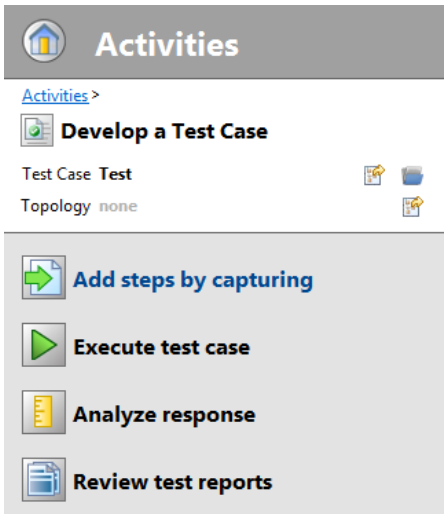
To execute a test case

1 After opening or creating the test case, on the **Develop a test case** activity page, click **Execute test case**. The **Execute Test Case** activity page opens.

2 The current test case is identified at the top of the page. If a topology or testbed file or parameters file is associated with the test case, then they are also identified.

- Click the filename to open the file in the appropriate editor
- Click the button next to the filename to change to a different file

3 To start execution, click . Use the following tools to control execution — you will find full details in "Debugging: Executing procedures, Pausing, stopping, and single-stepping" on page 417 and "Single-stepping through a test" on page 420.



Messages table

During execution, the **Messages** table displays the **severity** and **execution message** that is generated for each **execution issue**.

Tip To view additional information on each execution issue, open the Execution view. See "Execution view" on page 274.

- Icons represent the severity of the execution issues: **Information**, **Warning**, **OK**, and **Error**.
- Execution messages appears in the **Message** cell and describe execution events like test case pass/fail, results of analysis rules, execution pause/resume, and so on. Some execution messages are built-in (identifying the topology used for execution or generated by events — for example, "Execution started"); others are defined by the test case developer (generated by analysis rule actions — for example, "Only 3 ports of 4 are responding"). Execution messages also appear in the Execution view and Step Issues view. See "Execution view" on page 274.

To analyze a response to validates a value or store it into a variable

One of the most important functions of a test case is to analyze (validate) the responses to steps, and then, based on the results of the analysis, take some specified action (like set the test result to **Pass** or **Fail**, and add an execution message to the test report about why the test passed or failed, for example, "Procedure **clearStats** succeeded in clearing the stats" or "Configuring Router2 failed at port config step".

In iTest, ***analysis rules*** specify both how to analyze a response and the actions to take based on the analysis of the response.

To add an analysis rule to a test case step

Select the step in the Test Cas**e** editor and then click **Add Rule**. The **Analysis Rule** wizard starts, as described in "About the Analysis Rule wizard" on page 667.

You can either click Add rule... or click the down-arrow on the button.

- Click Add rule... to open the **Analysis Rule** wizard.
- Click the arrow to select one of the types of analysis rule from a drop-down list. This is a shortcut that bypasses the first page of the **Analysis Rule** wizard.

To edit an analysis rule

Select the step in the **Analysis Rules** table and then click **Edit**. The **Analysis Rule** wizard opens to a page that enables you to modify the rule as needed. See "About the Analysis Rule wizard" on page 667.

To delete a rule, select it in the list and click Delete

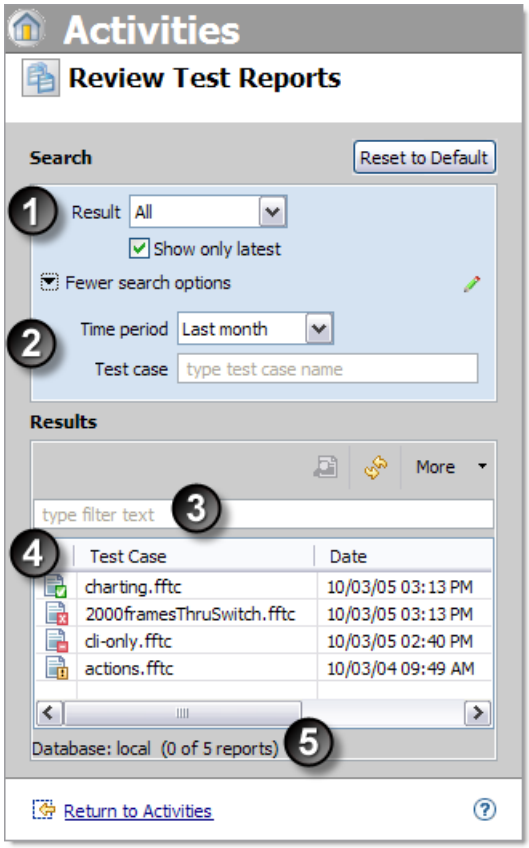
To view test reports for the current test case

Use the **Developing a test case** > **Review test reports** ("Reviewing test reports" on page 22) and fetch test reports for the current test case from the database.

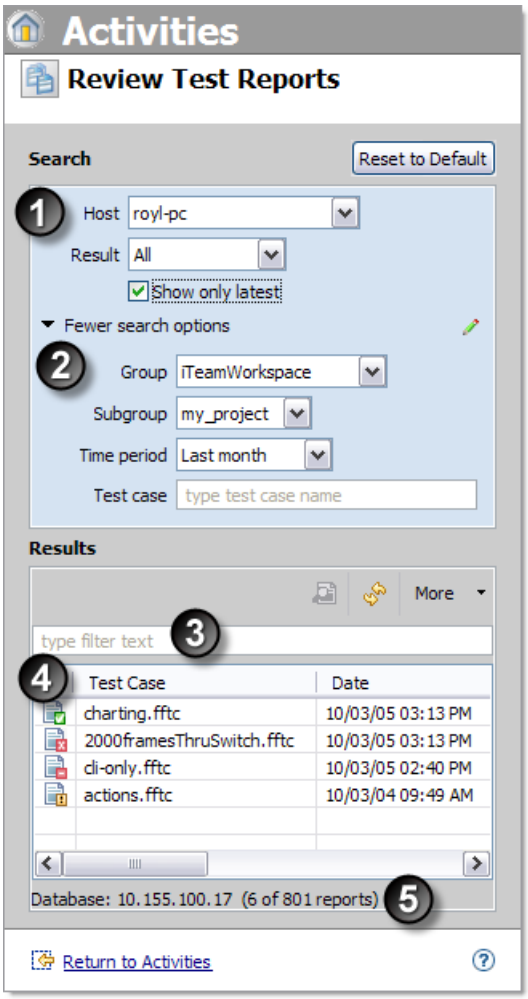
Reviewing test reports

Use the **Developing a test case** > **Review test reports** page to fetch test reports from the database for review. The view varies slightly to support the internal (default) or an external test report database:.

Internal database



External database



Note If you navigate to this page while developing a test case, then the search options relating to other test cases do not appear.

1 *Search the database*

Use these settings to perform a database search that populates the list of reports (that is, reports that meet the search criteria are listed).

Host	External database only. The search returns reports from the specified DBMS host.

Result	The search returns reports with the specified execution result: **Pass**, **Fail**, **Abort**, **Indeterminate**, or **All** (all of the result types).
Show only latest	If there is more than one test report for a test case that matches the other criteria, then fetch only the most recent test report of the o.

2 *Fine-tune the database search criteria*

Click the **More search options** arrow to access additional search settings. If you populate any of the following options, then the view is populated with the reports that meet both sets of search criteria. Specify any or all of the following optional search criteria.

Group	External database only. Optional. The search returns reports from the specified workspace
Subgroup	External database only. Optional. The search returns reports from the specified project
Time period	Specify the time period during which the test case was executed. Default: Last week.
Test case	Type the name of a test case document to fetch only reports for that test case. **Tip** See the **Show only latest** setting.

3 *Apply a filter to the reports listed on the Test Reports view*

Once the database search fetches the reports that meet the search criteria, you can further limit the reports that appear in the view by typing part or all of the text in any of the reported values, for example, the **Subgroup** name, the into the text box. Only reports for test cases that "pass the filter" now appear in the list. To reset so that no filter is applied, delete the text.

4 *List of test reports*

In the example, we see that four reports (one per line) matched the search criteria and passed the filter.

- Double-click a report to view it in the Test Report editor (described in "Test Report editor" on page 431).
- The icon in the first column represents the execution **Result** of the test case (**Pass** , **Fail** , **Abort** , or **Indeterminate**)
- Right-click a report to view a menu of options.
- For iTest's built-in database, you can specify a preference for how long to store test reports (that is, when to start discarding old reports). See "Setting preferences for Test Reports" on page 455.

Sorting reports in the view

By default, iTest lists reports in chronological order. You can click any heading to sort on a property of the report. For example, click the **Test Case** heading to sort in alphabetical order or reverse alphabetical order. Within each group of identically-named test cases, reports are sorted by **Timestamp.**

- Click the **Result** heading (the first column) to sort in the following order: **Pass**, **Indeterminate**, **Fail**, and **Abort**. Within each **Result** group, reports are sorted by **Timestamp**.

Result	Icon representing test execution **Result**: **Pass**  , **Fail**  , **Abort**  , or **Indeterminate** 
Timestamp	Date and time that the test case was executed (tests executed today display only the time).
Test Case	Name of the test case that executed to generate the report. To display the full URI, select the **Show Full Test Case URIs** option in the context (right-click) menu,
Result	Test execution **Result** in text form: **Pass**, **Fail**, **Abort**, or **Indeterminate**
Group	Optional. External database only. You specify that the **Group** value should appear in the list by setting a preference value. If you specify a value, on the **Preferences** page, then it appears in the **Group** column on the Review Test Reports activity page and on the Test Reports view. The value acts as a parent to the optional **Subgroup** value.
Subgroup	Optional. External database only. If you specify a value, then it appears in the **Subgroup** column on the Review Test Reports activity page and on the Test Reports view. The value acts as a child of the optional **Group** value.
Report ID	Unique auto-generated ID for the report. You can search on this value, open the test report using the ID, and access the ID using the **info reportId** command in a test case step.

Context (right-click) menu:

Open Test Report	Open the selected report in the Test Report editor (described in "Test Report editor" on page 431).
Delete Selected Test Reports	Use Ctrl+click and Shift+click for multi-select Alternatively, press the **Delete** key.
Save Test Report As	**Save the test report as HTML, text, or XML**: See "Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file" on page 451.
Quality Center	This option appears only if yo u use iTest with Quality Center. To start the **Publish Test Report to Quality Center** wizard, select one or more reports, right-click and select **Quality Center > Publish Test Reports to Quality Center.**
Show Full Test Case URIs	Alternate between displaying test case names and test case URIs in the **Test Case** column.
Open by Report ID	Specify the **Report ID** to open the selected report in the Test Report editor (described in "Test Report editor" on page 431). The **Report ID** appears in the list.

5 ***Source of the test reports and connection status***

This line displays the source of the test reports: the **Internal** (built-in) database in your current workspace (the default source) or an external database. For more information, see "Configuring Spirent iTest to save test reports to an external database" on page 447.

If there is an issue with the database connection, then the error message appears here.

CHAPTER 3

About the iTest Window

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

How the iTest window is organized

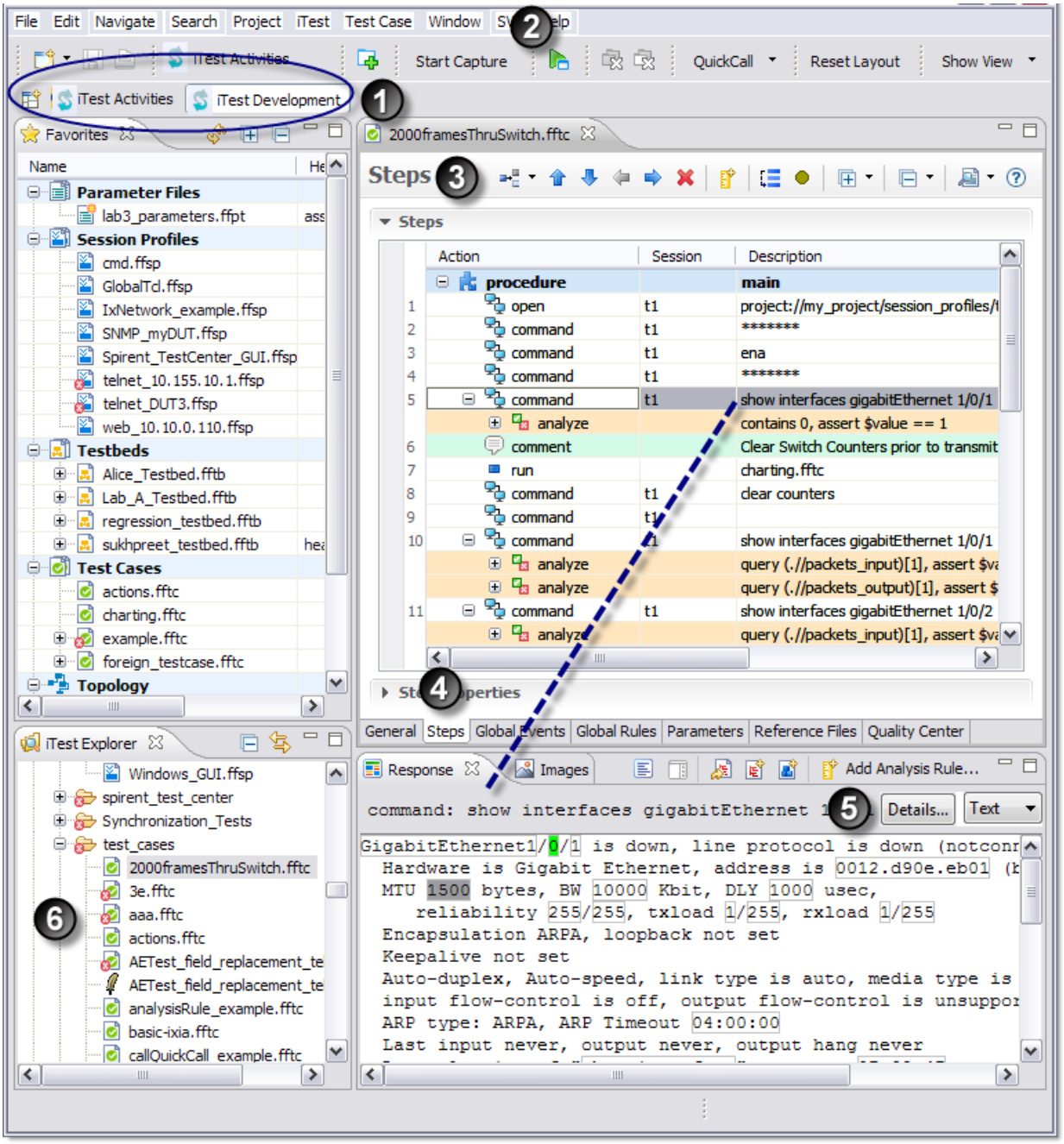
Because you can use iTest for a large variety of tasks, we provide a variety of built-in perspectives that organize the iTest window into practical arrangements of views and editors. For example, to make it easier to perform your most common activities, iTest provides the **iTest Activities** perspective, is described in Chapter 2, “Activity Pages”.

This chapter describes additional built-in perspectives.

Important Remember that a perspective is just a particularly useful arrangement of the available views and editors — no data changes when you change perspectives, the only changes are which data you are looking at and the way that the data is displayed.

How iTest perspectives are organized

Each iTest perspective organizes the iTest window into a practical arrangement of views and editors. Let's take a look at the **iTest Development** perspective:



Perspective controls. This group of buttons controls the current ***perspective***, the collection and arrangement of views and editors. We're viewing the **iTest Development** perspective. To switch to the **iTest Activities** perspective, click the **iTest Activities** button — a new arrangement of editors and views replaces the current arrangement. The buttons that

appear here are names of perspectives that you have used. Click to switch to any other perspective.

You can customize perspectives to suit your needs by moving, resizing, and opening or closing views and editors (don't forget to save your changes if you like them).

Main menu and toolbar. The menu and toolbar can change based on the current perspective or editor.

Editor. You work on any iTest document in an ***editor***. In the example, we opened the **2000framesThroughSwitch** test case document in the **Test Case** editor. In this editor, each line represents a step in the test. There are editors for topologies, for session profiles that define connection settings with devices, and for other types of iTest documents.

In an editor, you can type directly into text fields, select items from drop-down lists, and right-click to open a context menu of actions. The toolbar at the top of the editor enables you to take action on the currently selected item (in the example, we might add a step after the selected step, apply a breakpoint, skip the selected step, cut/copy/paste, and so on).

Pages that make up the editors. Editors can include several special-purpose pages that you open by clicking the tabs at the bottom. In the example, we clicked the **Steps** tab to work on the **Steps** page of the **Test Case** editor.

View. This is the **Response** view. Views display information on the document that you are editing and provide other supporting information.

In the example, we selected step 5 (the **show interfaces** command) on the **Steps** page of the Test Case editor. The **Response** view then immediately updated itself to display the command and the response to the command (from the most recent execution of the test case). iTest has used queries to return values of interest from the response (the values that match queries are surrounded by boxes, the value **0** that was verified by an "analysis rule" is highlighted in green).

On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form.

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.
- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").

- In the example, the **Response** and **Images** views are stacked. To bring a view to the front, click its tab.
- To display a view that does not currently appear, click Show View and select the view.
- Most views provide toolbars and context (right-click) menus for actions on selected items.

- Press **F1** or click ? for online help on the current view or editor.

⑥ The **iTest Explorer** view enables you to view and manage all documents that you create using iTest. We opened the **2000framesThroughSwitch** test case by double-clicking it. Right-click to open a context menu of actions like **Rename/Move/Copy/Paste/Delete/View Properties**. The Favorites view (directly above the iTest Explorer in the example) provides quick access to documents that you work with most often.

Important Typically, one iTest resource (the name for a iTest file) depends upon one or more other iTest resources. For example, a test case might depend on a topology, response maps in a response map library, session profiles, and so on.
As a result, when you **Rename**, **Move**, or **Delete** a iTest resource, multiple files will typically be affected. For details on how you can best rename, move, or delete a resource, see "Rebuilding project dependencies" on page 688.

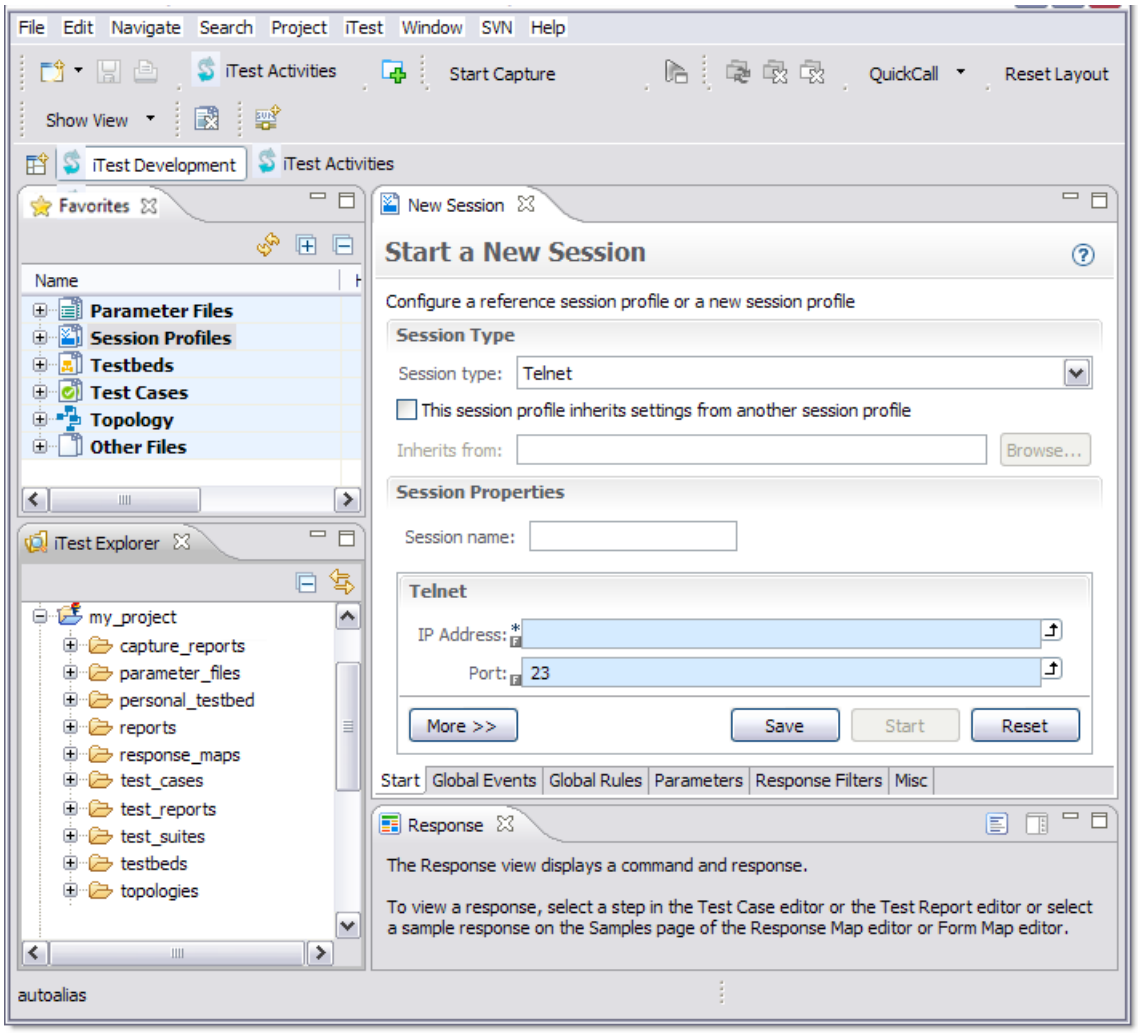
Tip Double-click a tab to maximize a view or editor. Double-click it again to minimize it.

The iTest Development perspective

Use the **iTest Development** perspective to perform manual tests on devices to capture steps for use in test cases. The following parts appear on the perspective:

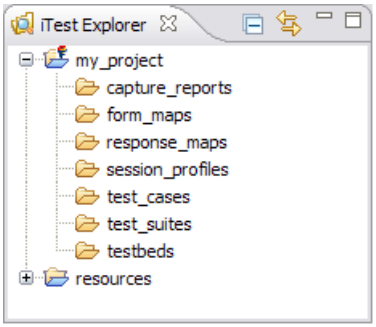
- The Favorites view gives you ready access to session profiles, test cases, and other files that you use often. You can place any sort of file in your Favorites view for easy access. You can even add non-Spirent files (such as spreadsheets or word-processor documents) to Favorites.
- The iTest Explorer displays all projects, libraries, folders, and files in your workspace.
- The **New Session** page is the first page of the Session Profile editor that you will use to configure and review session profile documents.
- The **Response view** displays the command and response for the currently selected captured item or test case step. The view makes it easy to review responses and to copy part or all of a response for pasting into other documents. (You can select captured items in the Capture

view or Capture Report editor. You can select test case steps in the Test Case editor and in the Test Report editor.)



iTest Explorer

The iTest Explorer displays all projects, libraries, folders, and files in your workspace.



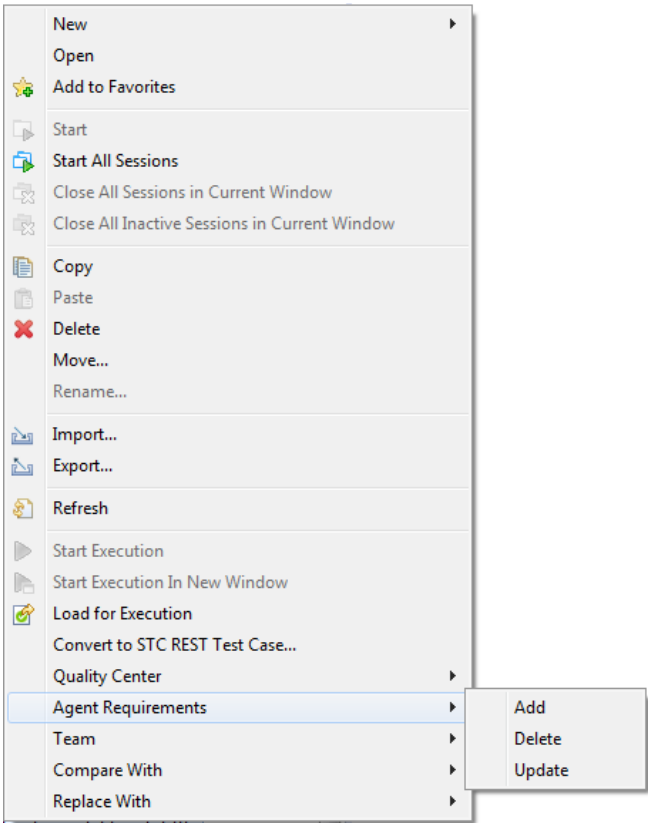
Important To view files and folders that are not in the workspace (for example, shared files that are held in itar files), use the External Projects view, as described in "External Projects view" on page 818.

- To open the iTest Explorer, click Show View in the main toolbar and then select **iTest Explorer**.
- One of the default projects in the workspace is named **my_project**. The icon looks like a hanging folder to indicate that it's a project that contains the folders.

  The other default project is named **resources** (for files like default SNMP MIB definitions and templates).
- When you first start, iTest supplies the folders within the **my_project** project that you see here. You can save iTest files anywhere you like, but here are the default locations for saving documents:
  - **capture_reports:** When you create a capture report, iTest suggests that you save it in the **capture_reports** folder.
  - **form_maps:** A library of form map documents that identify the actionable targets on Web pages
  - **response_maps:** A library of response map documents that simplify extracting data from a response
  - **session_profiles**: Session configuration settings saved as session profile documents
  - **test_cases:** Executable test cases
  - **libraries:** Quick call libraries or any other library files
- Double-click a document to edit it.
- Click **Collapse All** to collapse the directory tree for easier navigation.
- Select a test case and click **Execute** in the main toolbar to run it.
- In any folder, right-click a document to view a menu of options.

Performing Actions from iTest Explorer

The iTest Explorer Right-click action can apply to the entire project or the selected test cases/resources. This action is particularly useful if you have a large workspace with several projects and testcases within each project. For example, select the project, test case folder, or multiple test cases, select **Agent Requirements** and add a common requirement, update an existing requirement, or delete a previously applied requirement.



Folders that were created outside iTest

If you create a folder in the workspace using the operating system's tools (for example, Windows Explorer), the folder will not immediately appear in the CX40. To view the new folder, click in the CX40 and then press **F5**.

Executing a test case from the iTest Explorer

You can execute a test case from the iTest Explorer in any of the following ways:

- Select the test case. Click **Start Execution in New Window**
- Right-click it and select **Start Execution**
- Right-click it and select **Start Execution in New Window**

By default, when you start a session using a session profile from the Favorites view, the iTest Explorer, or the Session Profile editor, the session starts in a new session window.

Projects that are added to or deleted from the workspace

By default, when iTest starts, it imports any new projects that it finds in the workspace so that that they are visible in the iTest Explorer. This is important when you share a workspace under revision control — new projects are automatically imported.

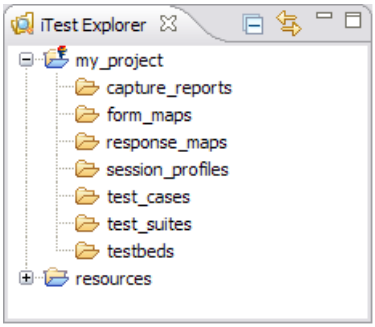
You can change the default behavior with a preference setting; see "Setting preferences for the workspace" on page 66.

Note Keep in mind that there can be some confusing interactions when, in addition to the projects in your workspace, you make use of projects in itar files. If you export a project to an itar and then use the iTest Explorer to delete the file, it is removed from the workspace but not from the file system. As a result, the itar'ed project might now appear on the External Projects view. The next time that you start iTest, however, iTest auto-imports the original project (because iTest discovered it in the workspace). The project now appears in the iTest Explorer once again. To use only the itar'ed version, you must use the operating system file management utility to remove the project from the workspace.

Projects

When iTest first starts up, the workspace includes two projects; one named **my_project** and one named **resources**. The projects appear in the iTest Explorer with icons representing hanging folders to distinguish them from folders. Projects contain folders and documents in any structure that you define.

- **my_project** is intended as your working project. You store all iTest files in **my_project**. You can add Response Map libraries and Form Map libraries to **my_project** as needed (default library folders are included in the default **my_project** structure).
- **resources** holds various built-in resource files, for example, the built-in SNMP MB files and XSLT template files.
- You can add, remove, or rename projects.
- Response Map libraries and Form Map libraries are actually projects and you manage them as you would any other project.



Working with projects

iTest allows you to create and work with as many projects as you need and you can add, delete, and rename the folders within any project.

Note Response Map libraries and Form Map libraries are actually projects and you manage them as you would any other project.

Tip You can set preferences for importing projects upon startup. See "Setting preferences for the workspace" on page 66.

Importing a project

To facilitate the use of approved shared files, your organization might ask you to import one or more existing projects. You import projects either directly as project files, as a project set file (.psf file) or by checking out project from the Subversion repository.

1 In the **iTest Activities** perspective, click the **Manage Workspace** activity.

2 Click **Import**. The **Import Project** wizard starts.

3 Select one of the following options:

- **Project set (.psf file)**. Your workspace may consist of several projects from one or more source control repositories. So that all can share the workspace, someone in your organization has exported the workspace into a project set. A project set is a text file that contains a pointer to each appropriate project.When you import a project set, the pointers are used to fetch the projects from the repository. Project sets can include any projects that are mapped to repository tooling that provides support for them, such as CVS.
- **Existing projects** in file system.
- **Checkout projects from SVN**. This option allows to import existing projects to the workspace from Subversion repository.

4 If you selected option **Checkout projects from SVN**, then click **Finish** and the standard Subversion checkout wizard opens (for more information please see Eclipse help on using Subversion). If you selected other options then browse to the appropriate location, select the project set or project, and then click **Finish**.

Project set: iTest fetches the projects in the project set from the repository. For some source control systems, you will be prompted to provide the authentication information (user name and password) for the repository (as it is not included in the project set).

Special case: Partial repository information

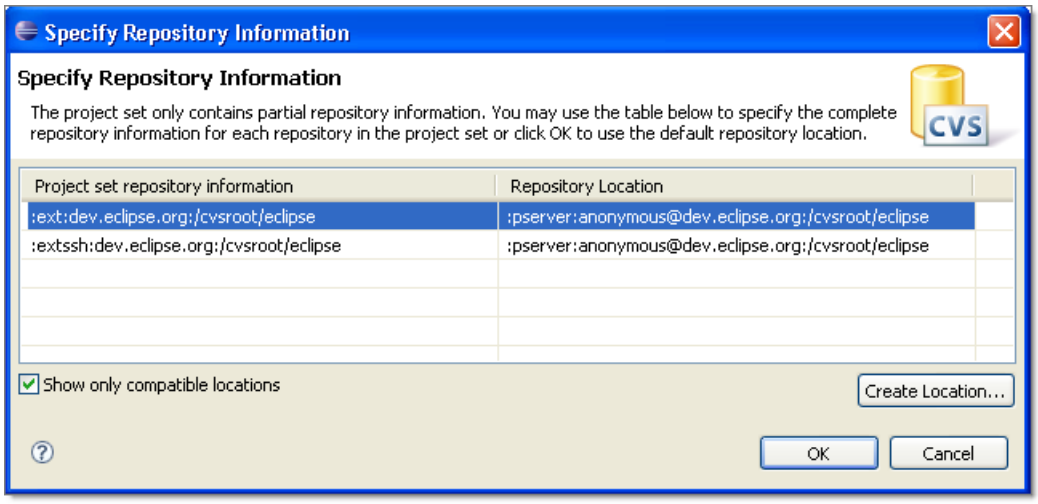
Note The following instructions apply only when the project set contains only partial repository information (not all locations are known).

The Specify Repository Information dialog box displays the following options for Repository locations:

- Use the original location that is displayed in the first row of the **Repository Location** column drop-down list.
- Click **OK** to choose a known repository location from the same drop down list (default).

- Create a new repository location by clicking **Create Location**.

You can fully configure the repository location, specify a user name, and change the connection method if required. For example, a project set file containing extssh connections can be used by a non-committer as they can reconfigure the repository location connection method on import to a pserver.



Existing projects: iTest adds a link to the projects into the current workspace.

Creating Reference Session

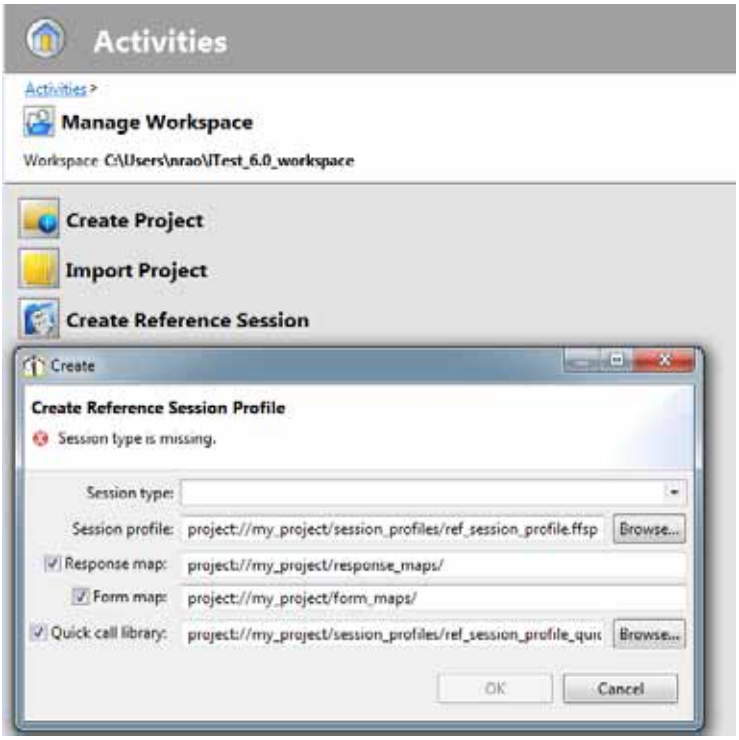
Use **Create Reference Session** on Manage Workspace (Activities > Manage Workspace) to create a profile that is referred by other session profiles.

To create a reference session while in the iTest Activities perspective

1. Click the **Manage Workspace** activity.
2. Click **Create Reference Session**.

3 In the **Create Reference Session Profile** wizard, select the **Session type.** The **Session type** and **Session profile** are mandatory. iTest provides default value/path for the **Session profile**.

Click browse to navigate to the target location, select the location and click **OK**. iTest populates the project name and folder path to indicate the location of the session profile.



4 Select optional **Response map**, **Form map**, and **Quick call library** options if required.

Click browse to navigate to the target location, select the location and click **OK**. iTest populates the required project name and folder path for **Response map**, **Form map**, and **Quick call library**.

Note iTest provides default values for **Session profile, Response map**, **Form map**, and **Quick call library** options associated with the **Session type** you selected. You may change the default value to any valid projects/folders/file names. An invalid input displays error message.

Creating a project

After you create a project, use the iTest Explorer to move it as needed.

To create a project while in the iTest Activities perspective

1 Click the **Manage Workspace** activity.

2 Click **Create Project.**

3 The **New iTest Project wizard** opens.

The following three options display:

- **Create default device project structure** – Selected by default

- **Create default automation project structure**
- **Empty project**

4 Select the required option to indicate the type of project to be created in the **New iTest Project** wizard and type the new **Project name** to be created.

5 Click **Finish**.

CAUTION Do not name projects **project**, **resources**, or **my_project**.

The table below lists the options on the **New iTest Project wizard** and the resulting folders created..

New iTest Project wizard option...	Resulting folders created in the new project...
Create default device project structure	Device_ project (for example) form_maps response_maps session_profiles
Create default automation project structure	Automation_project (for example) configuration parameter_files test_cases test_suites topologies
Empty project	Creates a new project with no folders.

To create a project while in any other perspective (Development or Activities)

1 Click **File > New** > **iTest Project**.

2 In the **New iTest Project** wizard, select the required option. See Step 3 above.

3 See Step 4 above.

4 Click **Finish**. See Step 5 above.

CAUTION Do not name projects **project**, **resources**, or **my_project**.

Deleting a project

In the iTest Explorer, right-click the project and then select **Delete**.

CAUTION: When deleting/recreating projects

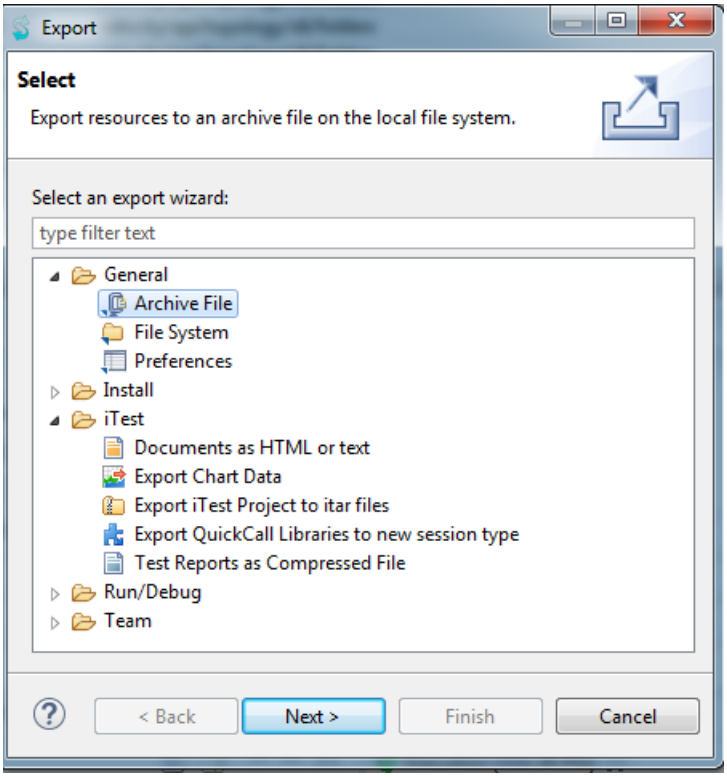
When you delete a project, all other projects in the workspace get rebuilt to add error markers to the projects that reference the deleted project. If you then create a new project with the same name as the deleted project, iTest does not perform another build (for performance reasons), and the error markers in the projects remain (incorrectly).

To avoid this possible problem, when you add a new project or delete a project, perform a clean build (click **Project > Clean**) on the affected projects to ensure that only appropriate error markers appear.

Sharing a project with another iTest user or with Customer support

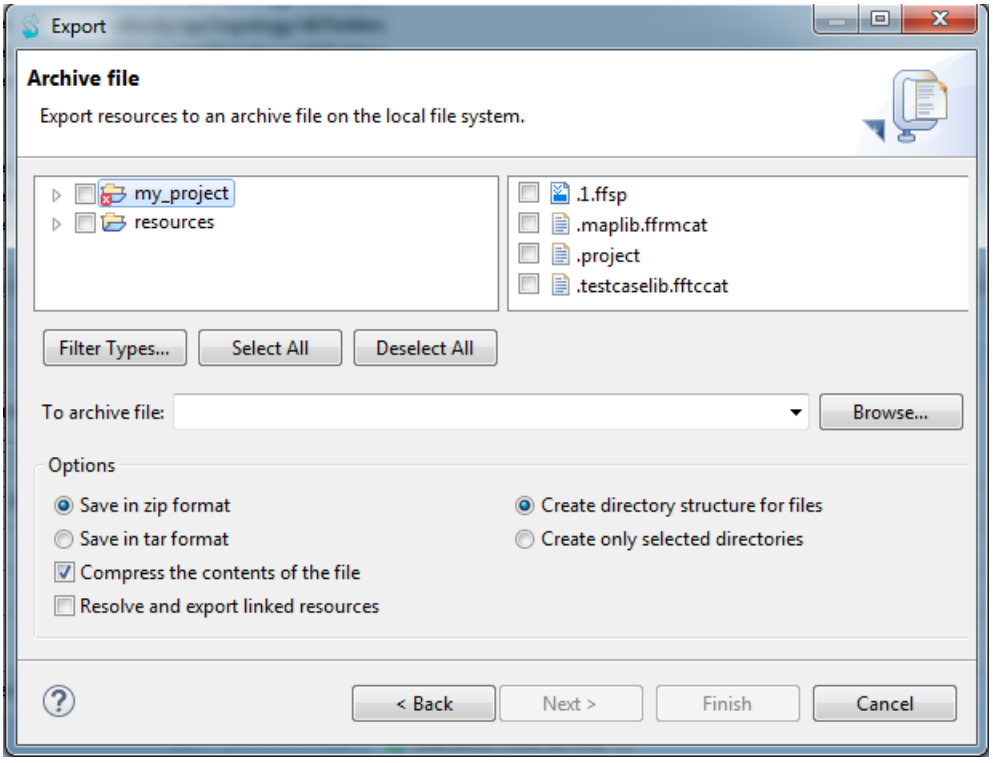
Step 1 First, Export the project to an archive:

1. Right-click the project and select **Export**. The Export wizard starts.
2. On the **Select** page, the **General > Archive File** option is selected. Click **Next**.



3. Click **Select All**.
4. In the **To archive file** text box, type and/or browse to the archive file name.

Note Do not name the project "**my_project**" or "**project**".



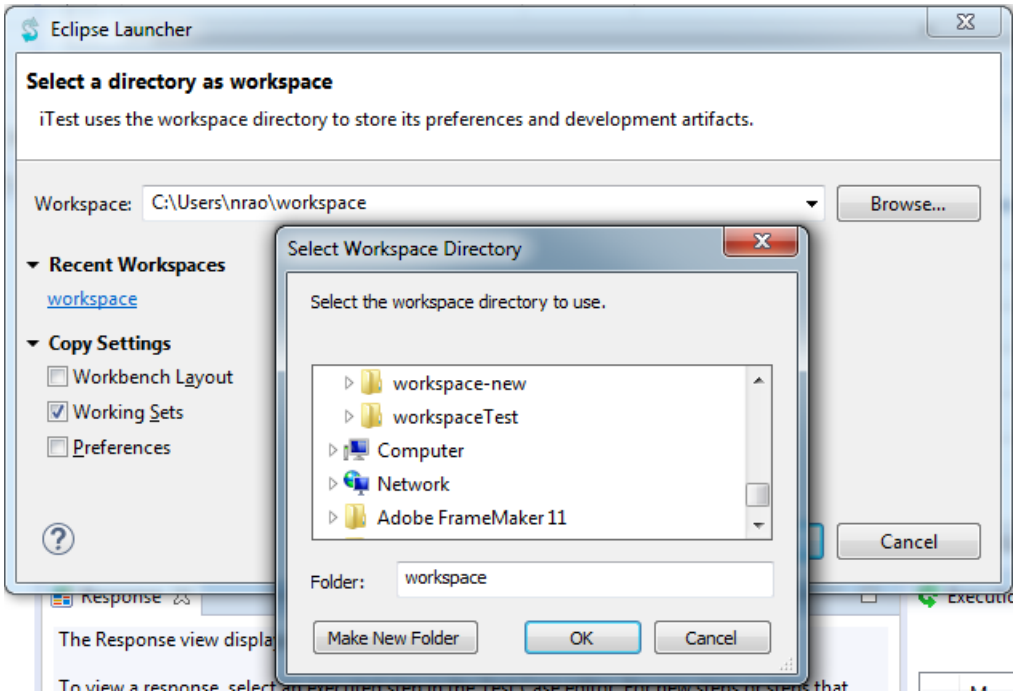
5 Click **Finish**.

Step 2 Now, your coworker can import the archive

To import the archive as a project:

6 Start iTest and create a new workspace: Click **File > Switch Workspace**.

7 In the Workspace Launcher, click **Browse**. Create a new directory and return to iTest.



8 In the iTest Explorer, click } to collapse all projects.

9 Delete the default projects to avoid duplicate names: Select (Ctrl-click) both the **my_project** and **resources** projects. Right-click and select **Delete**. On the **Confirm** dialog box, select **Also delete contents in the file system**.

The iTest Explorer will now be empty.

10 Click **File > Import**.

11 On the **Select** page, select **General > Existing Projects into Workspace**. Click **Next**.

12 On the **Import Projects** page, select **Select archive file** and then browse to the archive file.

13 Click **Finish**. The project is imported.

Specifying a default project for iTest

The ***default*** project is the project that is used when you create new files (for example, creating a new test case from capture or creating a new session profile). iTest ships with a default project of **my_project**, but you can change the default project to suit your needs.

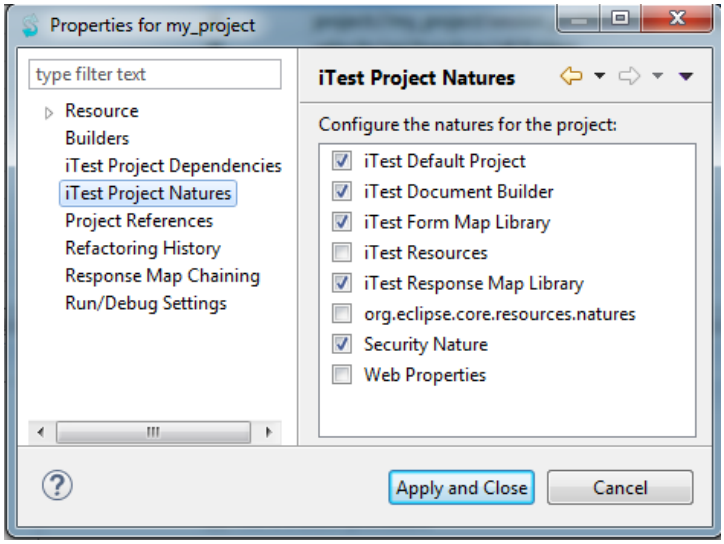
To specify the default project

The default project has a iTest Project *Nature* of **iTest Default Project**.

1 In the iTest Explorer, select the project.

2 On the **Project** menu, select **Properties**.

3 In the **Properties for [projectName]** dialog box, on the **iTest Project Natures** list, check the **iTest Default Project** check box.



The **Project Natures** are added by virtue of the Eclipse plugins installed on your system. The **Project Natures** allows you to tag a project as a specific type of project (e.g., iTest project) and indicate that a certain tool is used to operate on that project. Select the following options to specify the **Project Natures**:

iTest Default Project	Allows the framework to treat the files as required by iTest.
iTest Documentation Builder	Verifies files, checks for missing dependencies and other warnings that might exist.
iTest Response Map Library	Indicates that this project is a container of response maps that can be organized within a Response Map Library.
iTest Resource	Indicates a project that contains resources necessary for iTest to function. This includes templates for Test Report formats.
org,eclipse.core.resources.neture	This Eclipse resources plug-in provides services for accessing the projects, folders, and files with which you are working. For use iTest internal use.
Security Nature	For iTest internal use only. Security Nature is added by default to all new projects. **Selected:** iTest will automatically verify signatures of the test cases (.iTar files) and test report archives (*.fftz files) located in that project, creates problem markers and displays indicators accordingly. **Not Selected:** iTest will *not* automatically verify signatures of the .iTar and ..fftz files. **Note** It is recommended to use this only in conjunction with Spirent Support (**https://support.spirent.com/SpirentCSC/**).
Web Properties	For iTest internal use (this is an Eclipse artefact).

Note You cannot directly add options to the Project Natures. However, it is possible for you to install additional Plugins, which may add relevant Natures to the list of Project

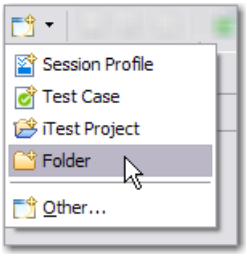
Natures. For example, if you add Python development plugins, relevant Natures may be added to the list of Project Natures.

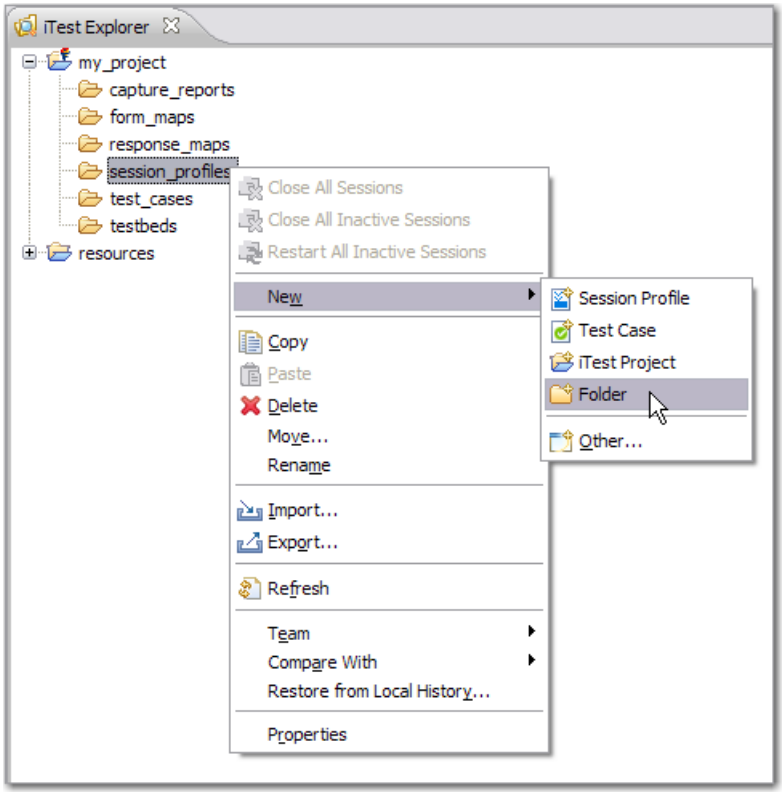
Working with folders

Once you create a folder in the iTest Explorer, you can add files to and delete files from the folder. You can create as many folders and levels of subfolder as you need.

Adding a folder

1 In the iTest Explorer, right-click the parent folder and select **New > Folder**. (Alternatively, click **File > New > Folder** or select **Folder** in the **New toolbar** drop-down menu.)





2 On the **New Folder** page, type the path to the parent folder or select it in the navigator.

3 In the **Folder Name** field, specify the name that should appear for the folder in the view.

4 For help on the **Advanced** options, click **Help** .

5 Click **Finish**.

Renaming a folder

Right-click the folder and select **Rename**.

Deleting a folder

Right-click the folder and select **Delete**.

Moving a folder

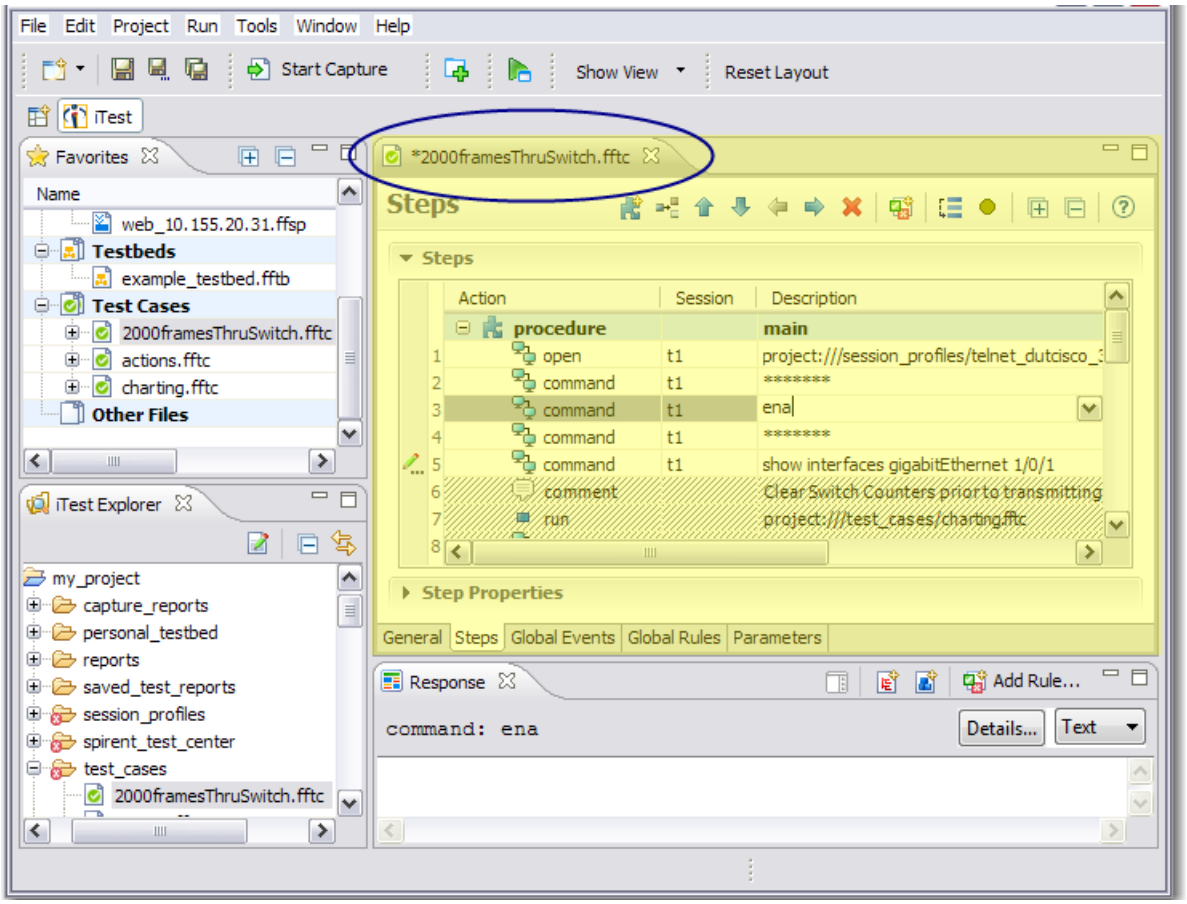
Select the folder and drop it in the new location.

Folders that were created outside iTest

If you create a folder in the workspace using the operating system's tools (for example, Windows Explorer), the folder will not immediately appear in the iTest Explorer. To view the new folder, click in the iTest Explorer and then press F5.

Editors

Editors are typically displayed in a tabbed window in the editor area (highlighted in the example).



Tip You can set preferences for editors. See "Preferences: Spirent > Editors" on page 863. Double-click the tab to maximize the editor. Double-click it again to minimize it.

Editors have menus

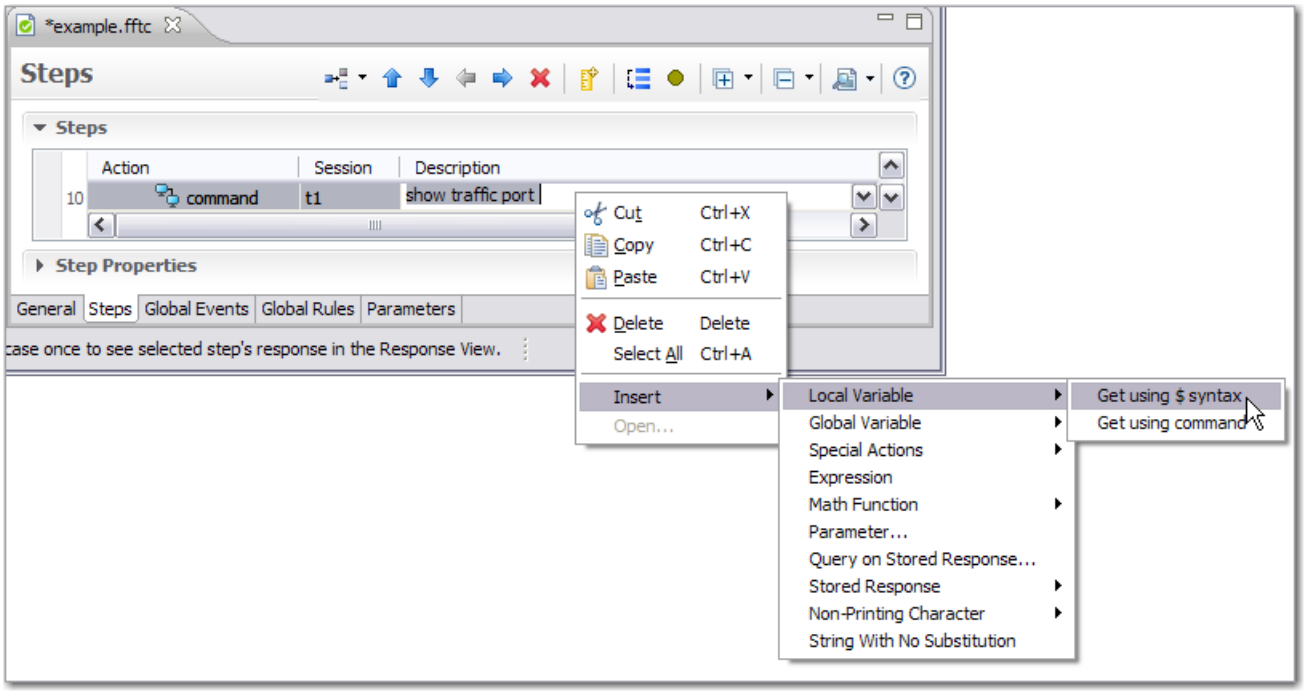
Whenever you open a iTest editor, iTest adds a menu to the main menu bar. For example, whenever the Test Case editor is selected, the **Test Case** menu appears in the menu bar. The options that appear in the menu vary depending on the editor page in use.

When in doubt, right-click

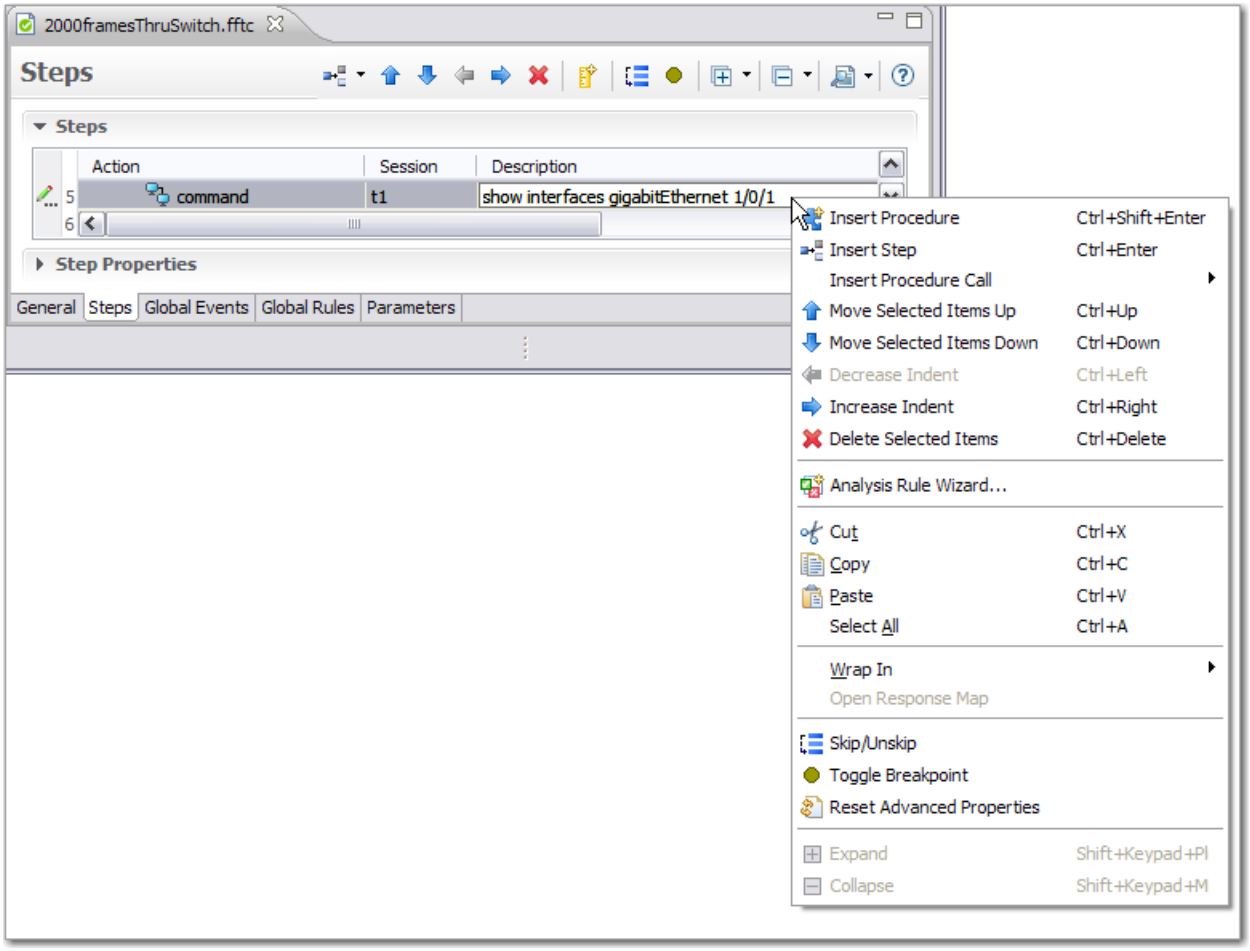
Most of iTest's editors and views include context (right-click) menus for common operations (most toolbar tools appear in the context menus).

For example, in the Test Case editor, there are two menus:

- To insert an item (for example, a variable), select the item to replace or place the cursor at the appropriate location and right-click:

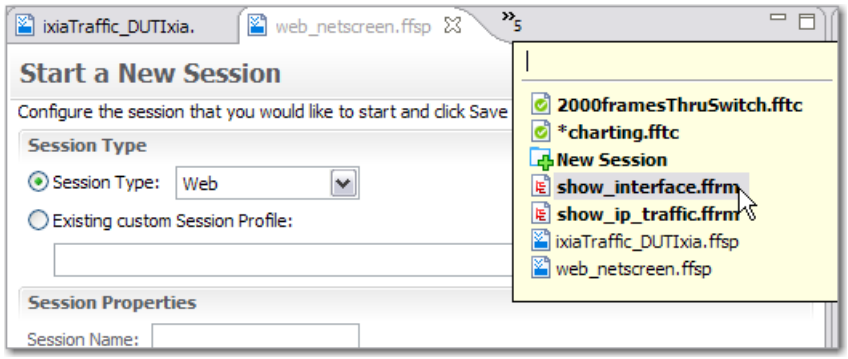


- To modify a step (skip it, wrap it inside a loop or comment, apply an analysis rule, and so on), select the step and right-click a cell:

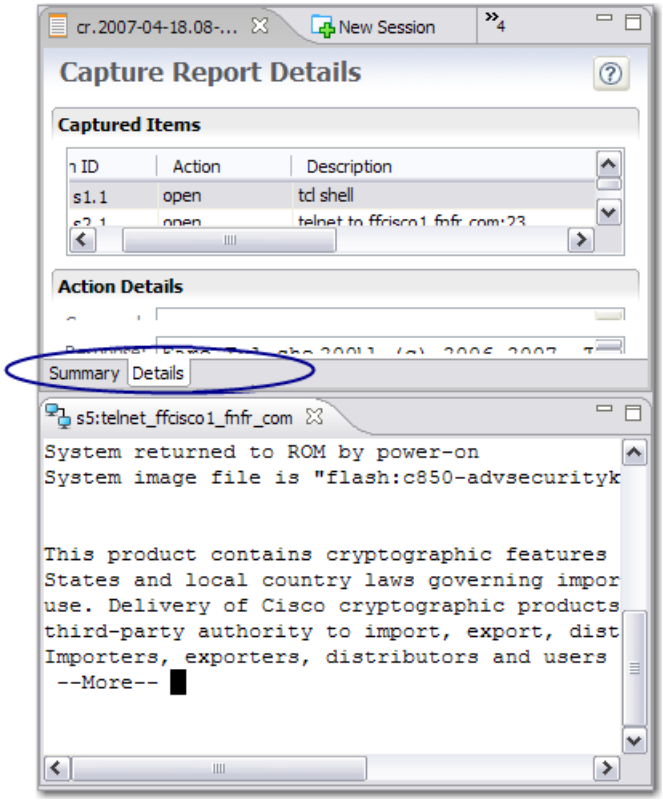


- Editors differ from views in that you can typically change the contents of an editor. For example, in the Test Case editor, you add and edit executable steps.
- Any number of editors can be open at once, but only one can be active at a time.
- The items in the main menu bar and main toolbar apply to the active editor.
- Tabs display the names of the documents that are currently open for editing (circled in the example).
- An asterisk * in the tab indicates unsaved changes.

- If there are more active editors that can be displayed, the select editor widget appears. The number indicates the number of editors that are not displayed. Click the widget to select an editor from the list.



- You can cycle through stacked editors using the back and forward arrow buttons in the toolbar and Ctrl+F6. The arrow buttons move through the last mouse selection points and permit moving through several points in a file before moving to another one. Ctrl+F6 pops up a list of currently selected editors, by default, the editor used before the current one is selected.
- By default, editors are stacked in the editor area (each newly opened editor is placed “on top of” the editor that is already open and each editor displays a tab at the top for access). You can choose to tile editors in order to view them simultaneously. In the example, the Capture Report editor (with the **cr.2007-04-18** tab) and the New Session editor are stacked and we have dragged a Telnet session window so that it is tiled below the Capture Report editor.



- Notice the two tabs at the bottom of the Capture report editor (**Summary** and **Details**). Some editors display tabs along the bottom edge to enable you to view pages that let you work on other aspects of the document being edited. In the example, the **Details** tab on the Capture Report editor is selected.

Launching editors

You can launch an editor in any of the following ways:

- Double-click a file in the iTest Explorer
- Right-click a file in the iTest Explorer or Favorites view and then select **Open**

Perspectives

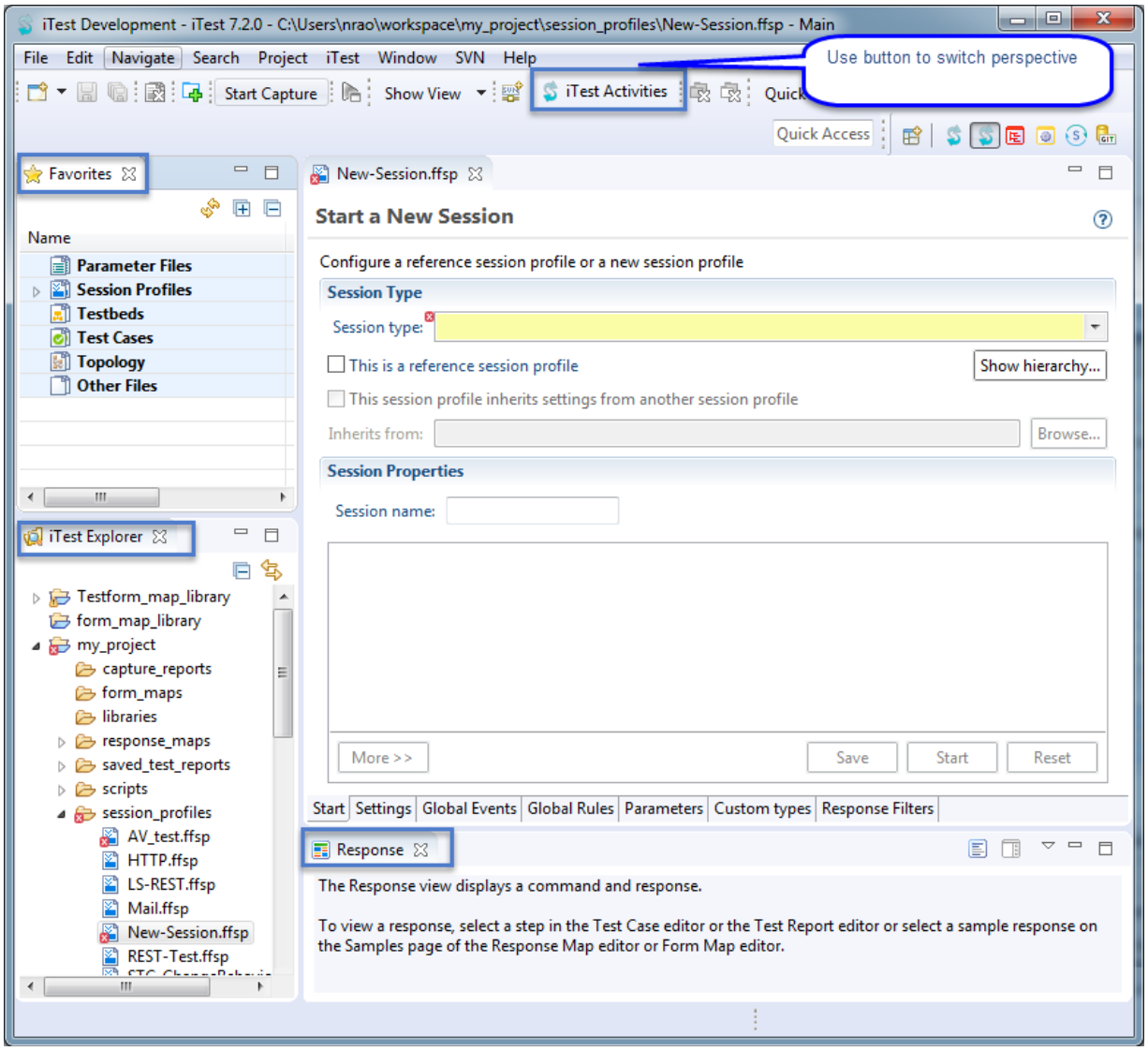
A *perspective* is the current layout of the iTest window—the arrangement of editors, views, menus, and toolbars that is optimized to help you get the current task done quickly and efficiently.

You'll switch between perspectives (that is, change the way iTest is laid out) depending on the task at hand. As you become a power user, you might even define custom perspectives to suit your needs. Switching between perspectives has no effect on iTest other than rearranging the windows and toolbars, so you are free to change perspectives any time you want.

Here is an example perspective (named **iTest Development**):

- You can see that the **iTest Activities** perspective is easily available at a button-click.
- The **Favorites** view, **iTest Explorer** view, **Response** view, and the New Session editor appear in this perspective. The editor appeared when we double-clicked a iTest document in the iTest Explorer.

- The views, editors, and toolbars that appear in a perspective are optimized to support the tasks that you perform. For example, the layout of the **iTest Response Mapping** perspective provides tools that speed up and support your work with response maps.



Tip You can set preferences for perspectives. See "Setting preferences for perspectives" on page 54.

Overviews of the default iTest perspectives

Here are descriptions for each built-in perspective. Select the perspective to view it while reading about the various views and editors that comprise the perspective.

iTest Development perspective

Use this perspective to perform manual tests on devices to capture steps for use in test cases. The following parts appear on the **iTest Development** perspective:

The Favorites view gives you ready access to session profiles, test cases, and other files that you use often. You can place any sort of file in your Favorites view for easy access. You can even add non-Spirent files (such as spreadsheets or word-processor documents) to favorites.

The iTest Explorer displays all projects, libraries, folders, and files in your workspace.

The blank open area (or the iTest Home page) on the upper right is an editor area that will be filled when you configure and review iTest test cases or session profiles.

The Response view displays the command and response for the currently selected captured item or test case step. The view makes it easy to review responses and to copy part or all of a response for pasting into other documents. (You can select captured items in the Capture view or Capture Report editor. You can select test case steps in the Test Report editor.)

iTest Debugging Perspective

During execution, when iTest encounters a breakpoint, it switches to the **iTest Debugging** perspective to facilitate single-stepping and other debugging tasks.

The Data view offers two modes:

- In the **Parameters** mode, you can:
  - View all parameters and both and local execution variables (as they appear in the heap)
  - View where they came from (in the **Source** column)
  - View current local variables for each thread (listed under the thread name).
  - Change parameter and variable values before executing a test case and while execution is paused. To change a parameter setting, select the value in the view and type a new value. (The **Description** text for each parameter is taken from the **Description** property, as specified on the **Parameters** page.)
  - View session profile parameter values for each session (listed under the session name).
- In the **Stack** mode, you can view variables for a stack frame in each thread.

The Threads view enables you to monitor detailed execution progress.

iTest Execution perspective

Use the **iTest Execution** perspective while you execute and debug test cases or replay captured sessions.

- Test Reports view: Whenever you execute a test case, iTest saves all of the steps and responses as a test report. Each test report relates to a single execution of a specific test case, so test reports are identified by the test case and the time of execution. The Test Reports view displays an ordered list of test reports and allows you to review, export, or delete them.
- The Execution view displays system-defined and user-defined **execution issues** that are generated during execution. Each issue has a **severity** (**Information**, **OK**, **Warning**, or **Error**) and one or more associated text **messages** (for example, “Port 4 not responding”). The view displays the step number for which each issue was generated and the source of the issue.

- The Response view displays the command and response for the currently selected captured item or test case step. (You can select captured items in the Capture view or Capture Report editor. You can select test case steps in the Test Report editor.) The view makes it easy to review responses and to copy part or all of a response for pasting into other documents.

iTest Expert Perspective

The Expert perspective is designed for the iTest power user. This perspective includes views used by iTest experts and excludes views that are less frequently used.

Use the **iTest Expert** perspective while you develop tests cases to share with other users via Git repositors. You add captured items to a new or existing test case, edit response maps, view validation of your changes, etc.

The views, editors, and toolbars that appear in the iTest Expert perspective are optimized to support the tasks that you, an iTest expert user, perform.

The iTest Explorer displays all projects, libraries, folders, and files in your workspace.

Capture view displays the captured items which you may select and add to a test case or create a new test case.

The Git staging and Git repositories views for working in your organization's version control system.

Git staging view shows the directory and file with which you are currently working.

Git repositories view displays the existing repositorees, add an existingrepository, clone a Git repository, or create a new Git repository.

The **Properties** view displays information depending on your selection, that is, whether you have selected a test case step or an item in the iTest Explorer (alphabetically). You may edit step properties using either the Step Properties section (within the Test Case Editor) or via the Properties View window.

The **Response** view displays the command and response for the currently selected captured item or test case step. (You can select captured items in the Capture view or Capture Report editor. You can select test case steps in the Test Report editor.) The view makes it easy to review responses and to copy part or all of a response for pasting into other documents.

Problems view: iTest performs validation on your changes to documents. When you take action that has the potential to cause problems (for example, during execution), iTest opens the Problems view and posts a message into the view.

Queries view: When iTest searches a response for a value that a test case step has requested, it uses an XPath query to search for the specified value in an XML-format structured data representation of the response text. XPath is the foundation on which response mapping is built.

Structure view dsiaplays a structured XML representation of every response. Use the Structure view to view the structured part of the response.

iTest Response Mapping perspective

You'll use the **iTest Response Mapping** perspective while creating and editing response maps using the Response Map editor and its associated views.

The iTest Explorer displays all projects, libraries, folders, and files in your workspace.

The Response view displays the command and response for the currently selected captured item or test case step. (You can select captured items in the Capture view or Capture Report editor. You can select test case steps in the Test Report editor.) The view makes it easy to review responses and to copy part or all of a response for pasting into other documents.

Step Issues view: While you work on a response map, the Step Issues view shows Errors, Warnings, and info messages associated with queries on the response in the **Response** view. The Step Issues view lists and describes unexpected items in the current sample, enabling you to “test drive” responses to determine whether named tokens (queries) work as you expect.

Problems view: iTest performs validation on your changes to documents. When you take action that has the potential to cause problems (for example, during execution), iTest opens the Problems view and posts a message into the view.

The **Error Log** view displays exceptions. The view comes to the front when an exception occurs. When working with Spirent Customer support, you can export and then email the contents of the log to help to resolve the issue.

Queries view: When iTest searches a response for a value that a test case step has requested, it uses an XPath query to search for the specified value in an XML-format structured data representation of the response text. XPath is the foundation on which response mapping is built.

The **Queries** view lists the XPath queries and their results for the response that is displayed in the Response view. (Queries can be defined in a response map or in local analysis rules. In addition, iTest auto-generates queries for structured responses like Web, SNMP, traffic generator devices, and XML.)

Structure view dsiaplays a structured XML representation of every response. Use the Structure view to view the structured part of the response.

iTest Session and SNMP perspectives

iTest Session perspectives open in a new window when you start a manual (interactive) session or iTest executes a test case. For CLI sessions, the perspective closely resembles a terminal client. The iTest SNMP Session perspective is an examples of a specialized perspective that opens an appropriate browser.

Git Perspective

The Git perspective is optimized for working in your organization’s version control system.

Git https://en.wikipedia.org/wiki/Git. See also Chapter 67, “Using Git in iTest”

CVS Repository and SVN Repository Perspectives

These perspectives are optimized for working in your organization’s source control environment.

About source control repositories

Source control is a mechanism for versioning and sharing files among a team. For a multi-discipline team, source control enables the automation team to share a variety of iTest files with testers who have historically performed their testing manually, enabling them to start automating test cases much more easily. For any type of team, source control provides a central, official location for all test assets. This eliminates versioning problems caused by email sharing ("which file is the latest?") and offers a central location for backup.

Popular source control solutions

- **CVS** (http://en.wikipedia.org/wiki/Concurrent_Versions_System)
- **Subversion, svn** (http://en.wikipedia.org/wiki/Subversion_%28software%29)

To import a project set file into iTest

In most cases, you should provide an Eclipse **Project Set File** (***.psf**) and credentials to a new iTest user, so they can import all of the projects needed to get started.

The iTest user then follows this procedure:

1. Click **File > Import**.
2. Navigate to **Team > Project Set File** and then complete the wizard.

To connect to a source control repository

If you do not provide a PSF file, then set up a connection to the source control repository. The team lead or automation team typically provides the repository URL to iTest users.

Team Synchronizing Perspective

This perspectives enables you to synchronize resources in the workspace with their associated remote CVS repository.

Switching between perspectives

To switch to a different perspective, click **Switch Perspective** and then select it from the list. Once you use a perspective, its button remains at the top so you can click to switch perspectives at any time.

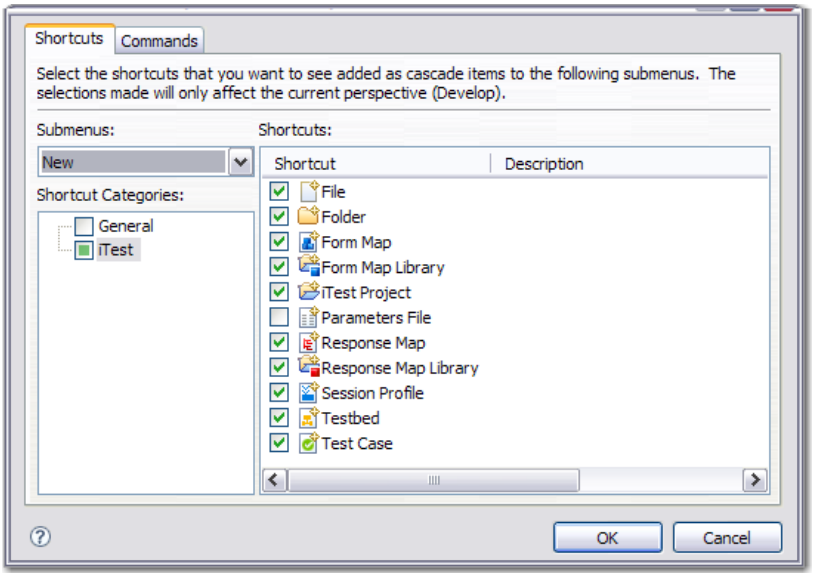
Customizing perspectives

You can customize any default iTest perspective to suit your needs. You can:

- Add or remove views or editors
- Drag views and editors to any location (use this feature to tile session windows)
- Resize view and editors

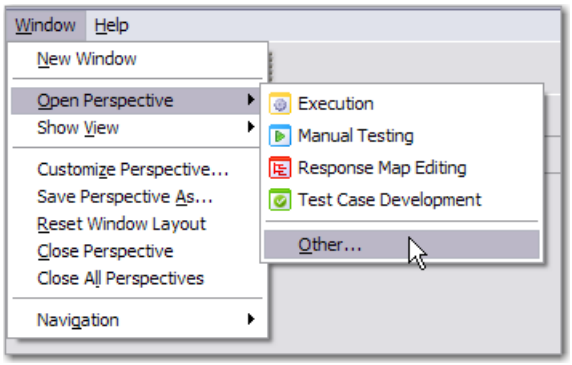
iTest keeps track of the new arrangement when you exit and restores it when you restart iTest. Instructions for saving your custom arrangement as a new perspective follow.

Alternatively, switch to the perspective that you want to customize. Click **Window > Customize Perspective**. On the Customize Perspective dialog box, specify the building blocks.



Saving a perspective layout

1. Click **Window > Save Perspective As**.
2. In the **Save Perspective As** dialog box, specify a name for the custom perspective and click **OK**. Now, when you open the new perspective, it is listed in the **Other** group.



Restoring the current perspective to its default appearance

To restore the default arrangement of editors and views in the current perspective, click Reset Layout.

Setting preferences for perspectives

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > General > Perspectives**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > General > Perspectives

Upon finishing the Add to Test Case wizard, switch to Test Case Development perspective	Once you click **Finish** on the Add to Test Case wizard, iTest can switch to the Test Case Development perspective. Default: unchecked
Upon finishing the Add to Test Case wizard, do not show the perspective switch dialog box	The perspective switch dialog box describes the reasons for switching perspectives and asks whether it is OK to switch. Default: checked
Upon finishing the New Response Map wizard, switch to Response Mapping perspective	Once you click **Finish** on the New Response Map wizard, iTest can switch to the **iTest Response Mapping** perspective. Default: unchecked
Upon finishing the New Response Map wizard, do not show the perspective switch dialog box	The perspective switch dialog box describes the reasons for switching perspectives and asks whether it is OK to switch. Default: checked
Upon finishing the New Test Case wizard, switch to iTest Development perspective	Once you click **Finish** on the New Test Case wizard, iTest can switch to the **iTest Development** perspective. Default: unchecked
Upon finishing the New Test Case wizard, do not show the perspective switch dialog box	The perspective switch dialog box describes the reasons for switching perspectives and asks whether it is OK to switch. Default: checked
When execution pauses, automatically switch to the iTest Debugging perspective	When an executing test case pauses, iTest can switch to the **iTest Debugging** perspective. Default: checked
When execution pauses, do not show the perspective switch dialog box	The perspective switch dialog box describes the reasons for switching perspectives and asks whether it is OK to switch. Default: checked
When executing from iTest Explorer, automatically switch to the iTest Execution perspective	When you execute a test case by selecting it in the iTest Explorer and then clicking  , iTest can switch to the **iTest Execution** perspective. You can execute a test case from the iTest Explorer in any of the following ways: Select it and then click **Start Execution in New Window**  Right-click it and select **Start Execution** Right-click it and select **Start Execution in New Window** Default: checked
When executing from iTest Explorer, do not show the perspective switch dialog box	The perspective switch dialog box describes the reasons for switching perspectives and asks whether it is OK to switch. Default: checked

Viewing the XML source of iTest documents

Advanced Users Only

To view iTest's XML representation of any iTest document (for example, Capture reports, test cases, and so on):

In the iTest Explorer, right-click the document and select **Open With Text Editor.**

iTest menus

File

New > iTest Project	Add a new project. See "Working with projects" on page 34.
New > Project	Add a new project, either an Eclipse project, or a iTest project. See "Working with projects" on page 34.
New > [iTest document type]	Add a new iTest document. See the appropriate section describing the various iTest document types.
New > Folder	Create a new folder. If you have not selected a parent folder in the iTest Explorer, then the new folder is added under the **my_project** folder. See "Working with folders" on page 43.
New > Other	Add a new document or other entity of the specified type. (A wizard opens to help you make the selection.)
Open File	Open the **Open File** dialog box to browse to and open a file.
Close	Close the currently active editor.
Close All	Close all editors.
Save	Save the document in the currently active editor.
Save As	Save the document in the currently active editor with a new name or path.
Save All	Save all open documents.
Revert	
Move	Move the items selected in the iTest Explorer to a specified folder.
Rename	Rename the document selected in the iTest Explorer.
Refresh	Refresh the currently active view or editor. **Note**: Because the Test Case editor auto-refreshes, the option is not applicable while editing a test case.
Print	Print the items selected in the iTest Explorer.
Switch Workspace > Other	Select another workspace in which to work. (A dialog box opens to help you make the selection.)
Restart	Exit the current instance of iTest (you may be asked to stop execution or to save documents) and then start a new instance of iTest.
Import	Import a document into the workspace.
Export	Export a document into another format or out of the workspace.
Properties	Display the properties of the document that is currently selected in the iTest Explorer
Exit	Exit iTest. You may be asked to stop execution or to save documents.

Edit

Undo/Redo	[All items in this menu perform the typical operations.]
Cut/Copy/Paste	
Delete	

Select All	
Find/Replace	
Add Bookmark	
Add Task	

Navigate

Go Into	
Go To	Back Forward Up One Level
Show In	
Next / Previous	
Last Edit Location	
Back / Forward	Move to the previous or next document that was being edited.

Search

Search	Perform a full-text search in the specified file.
File	Use the dialog box to define a detailed search of resources in the workspace.
Text	Workspace Project File Working Set

Project

Open/Close Project	Open or close the project selected in the iTest Explorer. You will be prompted to save modified documents.
Build All	See Chapter 30, “The iTest Builder”.
Build Project	See Chapter 30, “The iTest Builder”.
Build Working Set > Select Working Set	See Chapter 30, “The iTest Builder”.
Clean	See Chapter 30, “The iTest Builder”.
Build Automatically	See Chapter 30, “The iTest Builder”.
Properties	Display the properties of the project selected in the iTest Explorer

[iTest Document Type]

This menu item reflects the currently selected iTest document editor (for example, **Session Profile** or **Topology**). This menu item appears only when an editor is selected and includes options appropriate to the document type.

iTest

Run > Start Execution in New Window	Open another instance of iTest in a new window and execute the test there.
Session > Restart Inactive Sessions	For all open session windows that are not connected to the sessions, re-open the connection.
Session > Close All Inactive Sessions	For all open session windows that are not connected to the sessions, close the session windows.
Session > Close All Sessions	Close all session connections and their associated session windows.

Window

New Window	Open another instance of iTest in a new window
New Editor	For the current document in the current editor, open the document in another instance of the editor.
Open Perspective > [perspective_name]	"Perspectives" on page 48
Show View	Display the selected view on the iTest window. "Views" on page 287. **Tip:** It's quicker to click **Show View** on the main toolbar.
Customize Perspective	See "Customizing perspectives" on page 53
Save Perspective As	See "Customizing perspectives" on page 53.
Reset Window Layout	Change the current layout of editors and views to the default iTest layout. "Customizing perspectives" on page 53
Close Perspective	Close the current perspective (the previously-used perspectives is then used).
Close All Perspectives	Close all perspectives. A blank Eclipse window remains.
Navigation > Show System Menu	Display the context menu for the current view or editor (same as right-click).
Navigation > [all others]	Display, resize, or move to the selected view or editor on the iTest window.
Preferences	See Chapter 41, "Configuring iTest Preferences".

Help

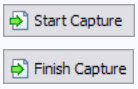
Welcome	Open the **Welcome** page to gain access to various resources.
Help Contents	Open the online help system in a new browser window.
Search	Perform a full-text search of the online help topics for the specified text string.
Dynamic Help	Open the help topic for the currently active view or editor.
Key Assist	Display the hot key assignments.
Tips and Tricks	View usage tips and tricks.
Cheat Sheets	Open and work through Eclipse cheat sheets.
Visit the Spirent Discussion Forums	Open the Spirent Discussion Forum for the latest forum, blog, video, and other helpful material from users of Spirent products and product experts.
View iTest Release Notes	Open a browser and view the up-to-the-minute release notes, system requirements, and "What's New" information.
Configure iTest Licensing	Open the **iTest Licensing** dialog box to review or edit the current licensing settings. Licensing is described in detail in the *iTest Installation Guide*, a PDF document that you can download from Spirent Support.
Note The next three options are recommended for use by admin personnel.	
Check for updates	Select to check for updates on Spirent update site as setup in "Preferences: Spirent > Software Update" on page 866.

Install new software	Select from the list of modular update releases (sessions, plug-ins: session type) that are independent of iTest general core/platform release cycle. **Note** On the **Preferences** dialog box, in the Preferences tree, click **Install/Update** to list the Eclipse settings, navigate to the A**vailable Software Sites** page and add/set up update site addresses as required.
iTest Installation Details	Select to view the current installation details: • Installed software: List of installed iTest software modules • Installation history: Date and time of installations • Features: lists of installed iTest features • Plug-ins: Installed iTest plug-ins • Configuration: Current iTest configuration In addition, you may view the error log and copy both the configuration and error log to clipboard to send it for support purposes.
About iTest	View information on iTest version, license agreement, and third-party software.

iTest toolbar

Main toolbar

The tools that appear in the main toolbar depend upon the current perspective. This table describes all tools that might appear.

	Create a new document of the specified type. **File** represents a text file.
	Save the current contents of the current editor as a file.
	Start / Stop the capture-to-test operation. See "Overview: Creating a test case" on page 125.
	Open/close the Session Profile editor (also called the **New Session** page)
	Start execution of the currently selected test case in a Session window.
 **Restart Inactive Sessions**	For all open session windows that are not connected to the sessions, re-open the connection.
 **Close All Inactive Sessions**	For all open session windows that are not connected to the sessions, close the session windows.
 **Close All Sessions**	Close all session connections and their associated session windows.

	Restore the default arrangement of editors and views in the current perspective.
	Display the specified view. Use one of the following methods to display a view: • In the main toolbar, click the -down arrow on the **Show View** button and select the view from the drop down list. If the view is not listed, then select **iTest** and select the view. • To select from a list of all available views, click **Show View** . On the **Show View** dialog box, select the view. • Select the view from the **Window > Show View** menu.

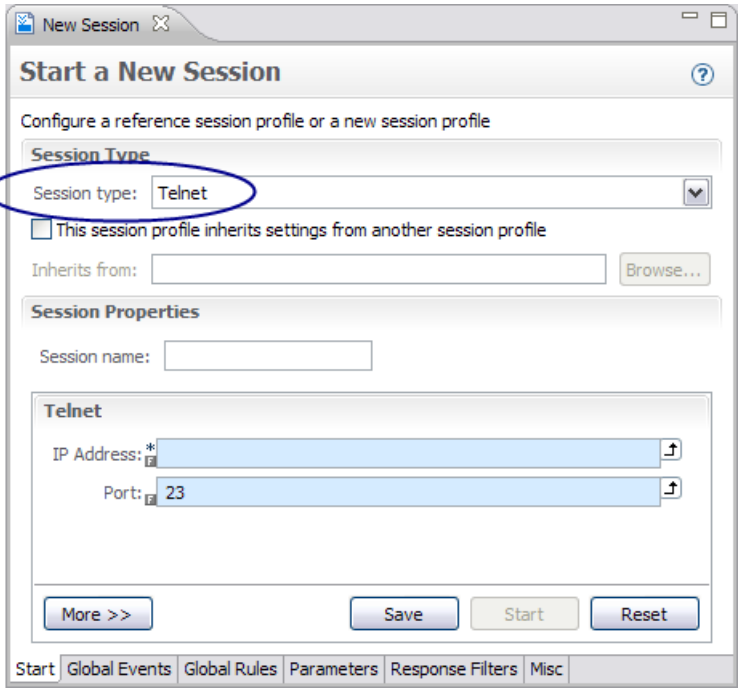
Keyboard shortcuts

You can use the following shortcuts in all editors:

Ctrl+Shift+L	Show all keyboard shortcuts in iTest
Ctrl+3	Open any iTest UI element (perspective, view, or editor) by typing its name.
Ctrl+Shift+R	Open a resource by name. In the dialog, specify the resource name. You can use wildcard characters * and ?
Ctrl+Shift+S	Save changes in all editors
Ctrl+Shift+W	Close all editors
Ctrl+Space	While working in the Test Case editor, display popup information to help you to enter appropriate contents for a cell.
Tab	Moves the cursor to the next field in the editor, if any.
Shift+Tab	Moves the cursor to the previous cell in the editor, if any.
Ctrl-Z	Undo
Enter	Confirms the changes to that cell and gets out of the edit mode.
F2	If a single item is selected, puts the cursor in the first editable cell in that item. If multiple items are selected, puts the cursor in the first editable cell of the first selected item.
F8	Sizes each column in the editor to fit the longest cell in that column.

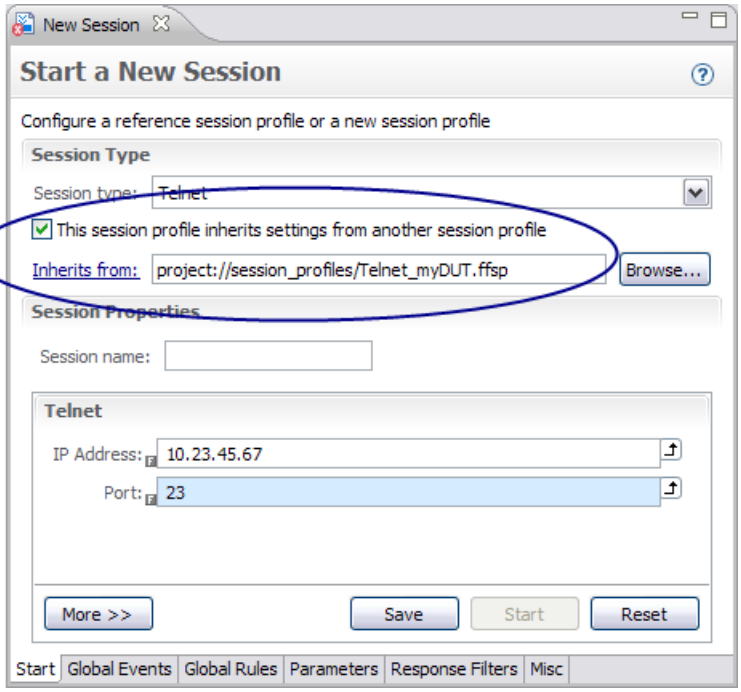
Property values: Inheriting settings

You'll notice while working in iTest that the **Inherit** button is attached to the right side of text boxes and drop-down lists for property settings. In addition, text fields, drop-down lists, and checkboxes are blue whenever they are set to ***inherit*** their setting (explained in the examples).



In the example, we are creating a session profile starting from iTest's default "Telnet" session type.

The fact that the field is blue and the up-arrow symbol is enclosed indicates that the property is currently ***inheriting*** its value from another session profile. In this case (a session profile that starts from the default Telnet **Session type**), the properties inherits values from the iTest default Telnet session profile. In the iTest default session profile for Telnet, the default **IP address** is **"**blank**"** and the default **Port** is **23**.



Here, we are defining a session profile that inherits its settings from a reference session profile.

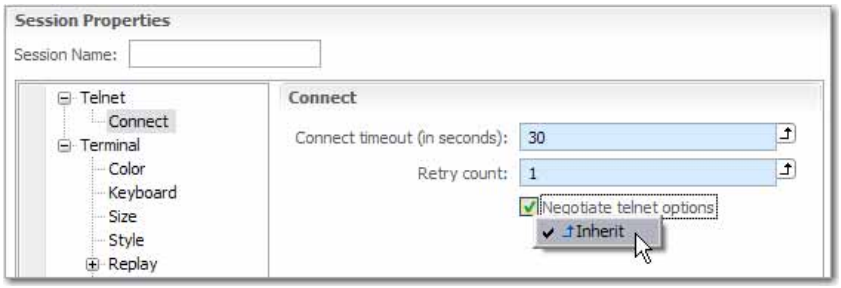
Again, the blue fields and enclosed arrows indicate that the properties are inheriting values, except in this case, they're inheriting values from the **Telnet_myDUT** session 'profile (a reference session profile that you can use as a template to speed up the process of defining a new session profile).

You can see that, in the **Telnet_myDUT** session profile, the IP Address is **10.23.45.67** and the **Port** is still the default **23**.

When you click the **Inherit** button or modify a setting, the field turns white and the up-arrow is no longer enclosed to indicate:
— The setting is no longer being inherited.
— You can edit the setting as needed.

In the example, we changed the **Port** value to **42**.

To reset the property to inherit its setting, click the **Inherit** button.



A blue checkbox indicates that the setting is inherited.

When you change the setting, the field turns white to indicate:
— The setting is no longer being inherited
— You can edit the setting as needed

Right-click to turn inheritance on or off.

About workspaces

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

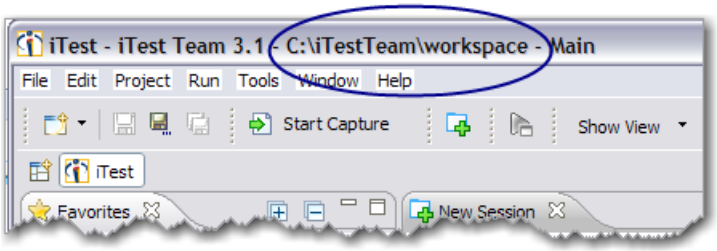
Your workspace is the folder on your file system that holds the projects in which you'll store all folders and files that you generate using iTest and the settings associated with your iTest environment.

When you first start using iTest, you work in the default workspace — the directory structure on your file system into which all iTest files are saved and from which all iTest files are opened. While working in iTest, you can add, move, rename, and delete folders just as you do in any file system.

A test case will almost always depend on other files – especially topologies or testbeds, response maps, and session profiles – but potentially other things as well (such as SNMP MIBs, and so on). To ensure that all such files can move around easily from computer to computer, iTest uses URIs rather than hard-coded paths — URIs that are always relative to the resource of interest. Many URIs will be relative to the root of the workspace. The fundamental idea is that when running a test case, if it depends on other files, the files need to be in a workspace (or in an itar file — more on them later).

Note Any preferences that you set apply to the current workspace only.

The path to the current workspace appears in the title of the iTest window:



Working outside of a workspace: itar files

iTest supports itar files to enable any iTest resource to work with any iTest file, even if it is not in a workspace. See "Exporting test cases and other iTest documents" on page 809.

The structure of the workspace

The contents of the workspace appear in the structured view that is displayed in the iTest Explorer.

Projects

Workspaces contain projects. Projects are not folders, they contain folders.

The default project is named **my_project**. When you open iTest, the iTest Explorer displays all working folders in **my_project**.

You can add, rename, or remove projects.

Response Map libraries and Form Map libraries

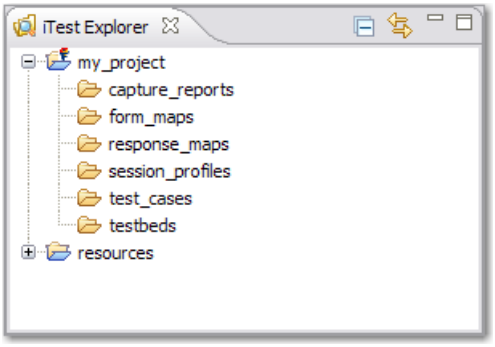
Response Map libraries and Form Map libraries are actually projects and you manage then as you would any other project.

Folders

Within projects, you can organize your working files (session profiles, capture reports, test cases, test reports, topologies or testbeds, and so on) into folders and subfolders. The default structure works for many users; you may want to move, add, delete, and rename folders and subfolders to suit your needs — you can organize the files into folders any way you want.

For example, you might further subdivide the session profiles in the **Sessions** folder into groups of sessions (for example, so you can start all sessions in a folder for particular tests or topologies or testbeds).

Here's the default directory (folder) structure under **my_project**:



- To open the iTest Explorer, click Show View in the main toolbar and then select **iTest Explorer**.
- One of the default projects in the workspace is named **my_project**. The icon looks like a hanging folder to indicate that it's a project that contains the folders.

  The other default project is named **resources** (for files like default SNMP MIB definitions and templates).
- When you first start, iTest supplies the folders within the **my_project** project that you see here. You can save iTest files anywhere you like, but here are the default locations for saving documents:
  - **capture_reports:** When you create a Capture report, iTest suggests that you save it in the **capture_reports** folder.
  - **form_maps:** A library of form map documents that identify the actionable targets on Web pages
  - **response_maps:** A library of response map documents that simplify extracting data from a response
  - **session_profiles:** Session configuration settings saved as session profile documents
  - **test_cases:** Executable test cases
- Double-click a document to edit it.
- Click **Collapse All** to collapse the directory tree for easier navigation.
- Select a test case and click **Execute** in the main toolbar to run it.
- In any folder, right-click a document to view a menu of options.

Default workspace

Microsoft Windows systems

The default workspace is **C:\Documents and Settings\[username]\My Documents\iTest**. You'll create your working folders in this folder. So, when you create a working folder called **load_tests**, for example, the path to the new folder is

```
C:\Documents and Settings\[username] \My Documents\iTest\load_tests
```

Linux and Unix systems

The default workspace is `/home/[username]/iTest`. You'll create your working directories in this directory. So, when you create a working directory called `load_tests`, for example, the path to the new directory is `/home/[username]/iTestload_tests`.

Switching to another workspace

You might switch to a different workspace in any of the following situations:

Your organization suggests that you use the default workspace as your personal workspace and share files with coworkers in a different “central” workspace.

You want to start with a completely new iTest environment

You want to store your test files in separate workspaces

To switch to a different workspace, follow the instruction in the Workspace Launcher.

Sharing a copy of your workspace

Important Zipping (compressing) a workspace into a zip file does not work. Instead, follow this procedure:

1. Click **File > Export > Archive File**, and select the portions of the workspace that you want to zip.
2. The recipient can use **File > Import > Archive File** to import the items into their workspace.

Making a copy of a workspace for Source Code Management (CVS)

To make a copy of a workspace to send to Spirent or to check into Source Code Management do the following:

1. Click **File > Export**.
2. Select **General > File System**. Click **Next**.
3. On the **File System** page, click **Select All**.
4. Browse to a destination for the workspace copy.
5. Click **Finish**.

A copy of the essential files of your test case, without history information, will be copied to the new destination. You can then check it in.

Setting preferences for the workspace

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > General > Workspace**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > General > Workspace

Automatically import missing projects into workspace on startup	**We strongly recommend that you do not change this setting.** If checked, then, when iTest starts, automatically import missing projects into the workspace. This setting is important when you share a workspace under revision control; new projects are automatically imported, avoiding confusion. **Note**: If you have specified the root of a file system as your workspace (strongly **not** recommended), then start iTest using the following command: itest --noImport iTest starts and deselects the **Automatically import** setting. You must then use **File > Import** to import the projects that you need. Default: checked

Working with Spirent documents

Spirent documents appear in the iTest Explorer under one or more folders. All Spirent documents use a filename extension that starts with the letters **ff**. For example, Capture report files use a **.ffcr** filename extension.

Adding a document

Click **File > New > [document type]**

Renaming a document

Right-click the document and select **Rename**.

Deleting a document

Right-click the document and select **Delete**.

iTest filename extensions

iTest uses the following filename extensions:

Document type	Filename Extension
Capture report (from iTest Personal)	**ffcr**
Capture report (zipped)	**ffcz**
Form map	**fffm**
Job definition	**ffjd**
Log file	**log**
Parameter file	**ffpt**
Preferences file	**prf**
Procedure (from iTest Personal)	**ffpr**
Properties file	**properties**

Response map	**ffrm**
Session profile document	**ffsp**
Test case document	**fftc**
Test report (zipped)	**fftz**
Test suite	**ffts**
Testbed	**fftb**
Text file	**txt**
Topology	**tbml**
XML document	**xml**

URIs

iTest uses URIs to refer to files. In iTest, when a document contains a reference to a resource, the reference is stored as a URI. The URI is used to find the referenced resource.

Formats of commonly used URIs

project://	Relative to the specified project. Format: **project://project_name/directory_name/subdirectory_name/filename** Relative to the current project. Format: **project://directory_name/subdirectory_name/filename**
file:/	Relative to the root of the file system. Format: file:/subdirectory_name/filename.fftc
Relative to the current file	Relative to the current file (typically a file in the same directory or in a subdirectory). File-relative URIs use no slashes after the colon. For example, **subdirBelowMe/filename.fftc** means a reference to a file called **filename.fftc** in the subdirectory **subdirBelowMe** that is under the directory where the current file is located. **../subdirNetToMe/filename.fftc** means a reference to a file called **filename.fftc** in the subdirectory **subdirNetToMe** that is at the same level as the directory where the current file is located.
. **(the current URI)**	Use " . " for a URI that is pointing at the same directory as the current URI's directory. This is similar to directory syntax in operating systems where . and ./ refer to the current directory and ../ refers to the parent directory.

Example URIs

Procedure call in a test case

In a test case, we find a **call** step that calls a procedure named **initializeSwitch**

The URI in the **Description** cell for the step is:

project://my_project/test%20cases/framesThruSwitch.fftc#initializeSwitch

- The project is named **my_project**

- The folder is named **test cases** (notice the space between **test** and **cases**)
- The name of the test case file is **framesThruSwitch.fftc**
- The name of the procedure being called is **initializeSwitch**

Testbed URI that specifies a device

project://my_project/testbeds/testbed.fftc#device=deviceName

Relative URIs

The general forms are:

fileName.extension

../fileName.extension

folderName/fileName.extension

URIs support relative references. For example, in an **open** step, the URI could be **../session_profiles/my_profile.ffsp**

Using relative URIs is good practice because you can move and/or rename things with minimal impact. For example, if you have a response map for a particular step in a test case, you could place the response map file in the same directory as the test case. In the step properties, use **my_local_respone_map.ffrm**

URI syntax

URIs cannot contain spaces, and therefore spaces in folder names and filenames are replaced by **%20** in the URI.

iTest applies the following encoding transformations to paths:

- The space character is converted into **%20**
- Alphanumeric characters **a** through **z**, **A** through **Z**, and **0** through **9** are unchanged
- Special characters **.**, **-**, *****, and _ are unchanged

Specifying how much memory to allocate to iTest

To specify the amount of memory allocated to iTest:

In the iTest.exe installation directory, edit the **iTest.ini** file and modify the following lines:

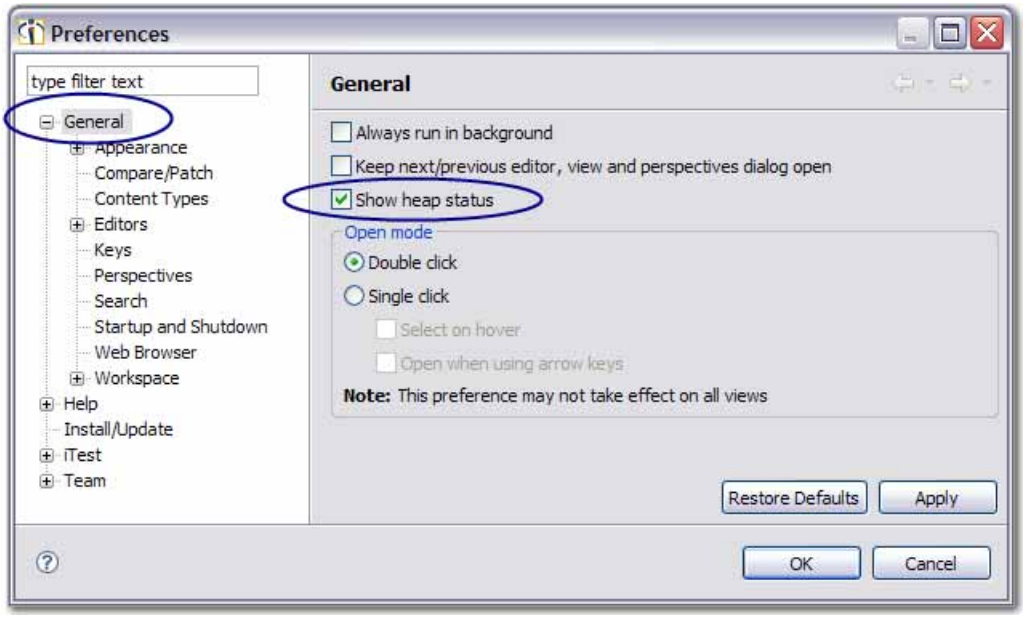
```
-vmargs
-Xmx512M (this is the default —512 MB memory)
```

To increase to 1024MB, for example, change the last line to `-Xmx1024M`

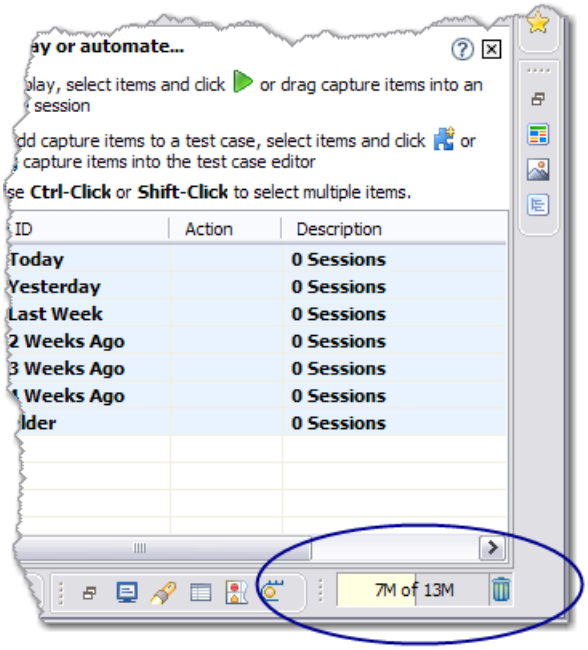
Monitoring iTest's memory usage

1. Click **Window > Preferences**.
2. On the **Preferences** dialog box, click **General**.

3 Click the **Show heap status** check box.



Heap utilization information will now appear in the bottom right of the iTest status bar.



CHAPTER 4

Session Profiles

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Session profiles: Session configuration settings

The collection of configuration settings that enable you to start a session is called its ***session profile*** (required settings like device IP address or hostname and port number, and optional settings like terminal text color, how to handle "more" line-continuation prompts, and so on). Each session profile is a document with file extension **.ffsp**.

Session profiles are used both to start manual (interactive) test sessions and to open sessions during automated testing (sessions can be with devices or applications). Each profile is of a particular type (Telnet, SSH, SNMP, Web, and so on) and a profile can inherit settings from another session profile.

Note Session profiles define how to *start* a new session (open a connection) and (except for knowing which prompts, break characters, or command-completion characters to expect) do not store any information about what occurs *during* a session (commands and responses, for example). Actions that occur during interactive sessions appear in the Capture view and can be saved as steps in test cases or stored in Capture reports.

About creating session profiles

- You create and edit profiles in the Session Profile editor.
- iTest provides a large number of default session profiles that you can use as the basis for your custom profiles.
- Once you create a profile, you can use it to start any session — you do not create a new session profile every time you start a session.
- You can use one session profile as the basis when creating a slightly different profile. This is called "using a reference session profile".

Reference session profiles

You will use a *reference* session profile to store the common configuration settings for a class of similar devices. When you base profile B, for example, on reference profile A, then profile B inherits all of its property settings from profile A. You can then make minor changes to profile B and it is ready to use.

For example, all RX5000 routers share most settings but each has a unique IP address. You create a single reference session profile that you can use to start a session with any RX5000. Whenever you need to configure a session connection with any RX5000 device, you can use the reference session profile as the starting point and then modify the IP address and perhaps some other settings and be ready to go.

Important It will save you a lot of time to use a reference session profile as the template when you add a new session profile configuration. You invest the time in maintaining the reference session profile with appropriate settings and (most importantly) the definitions of prompts that the session can return. Then, to add the new profile, you identify the reference profile, make a few minor changes, save it with a new name, and you are ready to go.

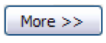
Using reference session profiles in testbed definitions

While an **open** step in a test case can refer to a session profile to start a session, we recommend that, instead, the step should refer to a device in a topology or testbed file. The testbed file lists all of the devices and session types that will be used while the test case executes. Because each device definition in the testbed refers to a reference session profile, common properties are stored and maintained centrally — you benefit from the ongoing improvement to the session profile definitions and avoid the debugging headaches associated with trying to keep a lot of similar files in synch. For example, in the device definition for the particular RX5000 at 12.34.56.78, you would set the **Inherit from** property to the reference session profile and then specify an **IP address** property setting of **12.34.56.78**.

About property settings

This topic describes the property settings that you can configure for sessions.

Device definitions, session definitions, and session profile documents supply the configuration settings that tell iTest how to open a connection with (launch) a session. For basic information on configuring a device or session profile, see "Session profiles: Session configuration settings" on page 71.

- On the **Connect to Devices** activity page, when you double-click a session type or click **Edit**, the properties appear on the **Start Session** dialog box
- On the **Topology** editor, when you select the session on the **Session** tab in the **Properties** view and click **Edit**, the properties appear on the **Edit Session Profile** dialog box
- On the **Testbed** editor, the properties appear in the **Device Properties** section
- On the **Session Profile** editor, the first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Note If you have already saved a device or session profile document with the appropriate settings, then you do not need to configure another document to start an interactive session. Instead, you can quickly start a session using one of the following methods:

- On the **Connect to Devices** activity page, select a session and click **Edit**
- In the Favorites view, expand the appropriate topology or testbed and double-click the device
- In the Favorites view, double-click the existing session profile

- In the iTest Explorer, right-click the existing session profile and select **Start**

Tips Here are some ideas that apply to any session profile:

- Remember that you can use field replacements to provide values for most properties in the session profile (properties with the F indicator). For example, you can specify the **IP address** property dynamically at runtime by passing a parameter value to the session. Let's say that the parameter is named **ip_addr**. In the session profile, for the **IP address** property, you would type: **[param ip_addr]**. See Chapter 28, "Field Replacements".
- For an **open** step in a test case, you have the option to override any of the property settings so that all steps in the session use the new property settings. Change any of the properties for the **open** step that appears in the **<sessionType> Session Properties** section. See "Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step" on page 183.

Defining a session profile (configuring the session settings)

To define a session profile, you first specify the type of session (one of: Telnet, SSH, SNMP, Web, or several others) and then specify settings that enable iTest to start the session (for example, IP address and how to handle "more" line-continuation prompts).

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

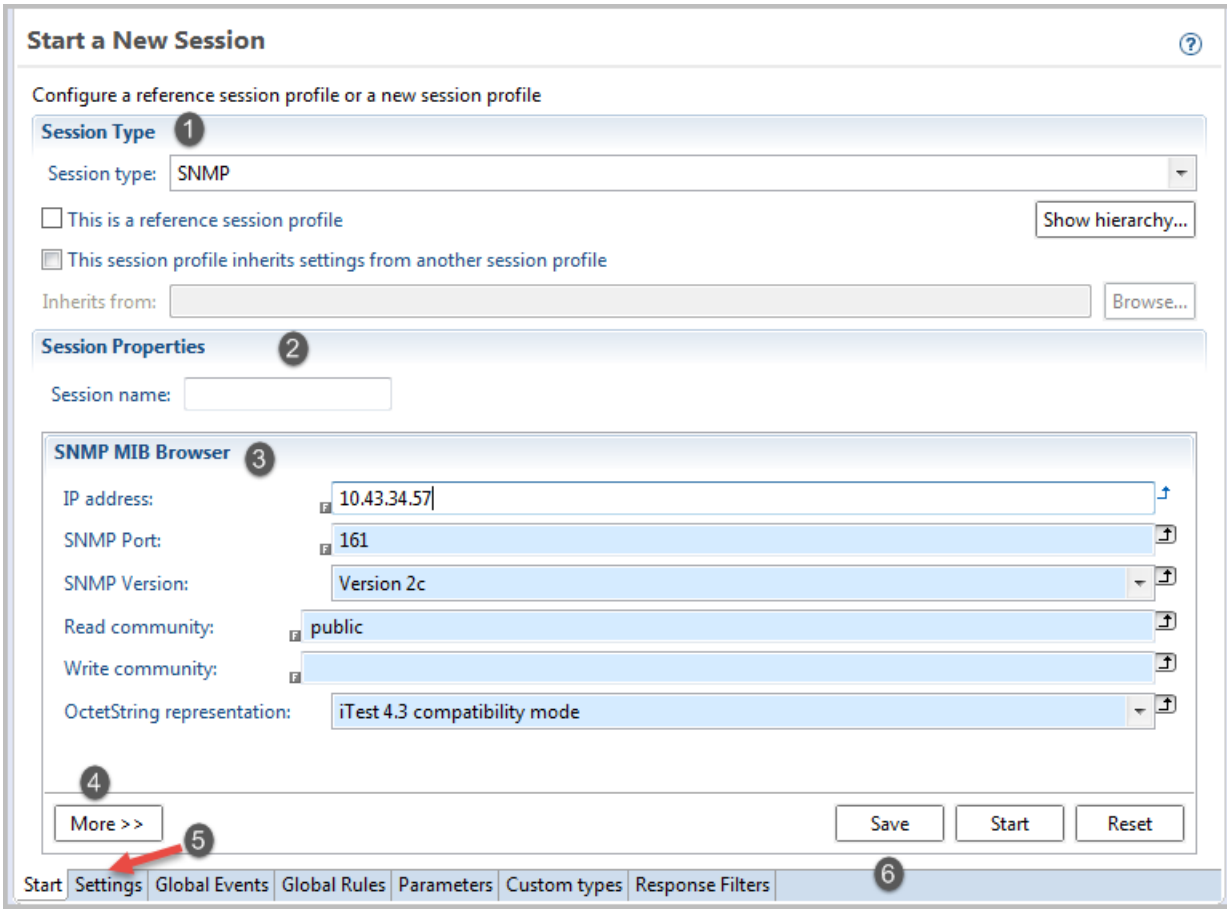
You create and edit session profiles using the Session Profile editor (also called the **New Sessions** page). The best way to define a new session profile is to start from a reference session profile and edit settings as needed for the new session.

To define a session profile

This example describes the **Start** page of the Session Profile editor. Each of the other editor pages is described in turn.

Tip Whenever the Session Profile editor is selected, the **Session Profile** menu appears in the menu bar. The options that appear in the menu vary depending on the editor page in use.

Click  to open the **Start** page of the Session Profile editor.



1 **Session Type:** There are three ways to start defining a new session profile:

- Inherit all property settings from the default iTest template for a session type. Specify the **Session type** (**Telnet**, **SNMP**, **Web**, and so on).
- Inherit all property settings from an existing session profile — either a reference session profile or any other session profile. This is the most common and the most powerful way to define a session profile, as described in "Reference session profiles" on page 71.

To specify that **this is a reference session profile**:

a Specify the session type (we selected **SNMP** in this example).

b Check **This is a reference session profile**.

From now on, when you are defining other session profiles, the reference session profile will appear in the drop-down **Inherits from** list on the Session Profile editor's **Start** page.

To specify that this session profile should inherit settings from a reference session profile:

a Specify the session type (we selected **SNMP** in this example).

b Check the **This session profile inherits settings from another session profile** check box

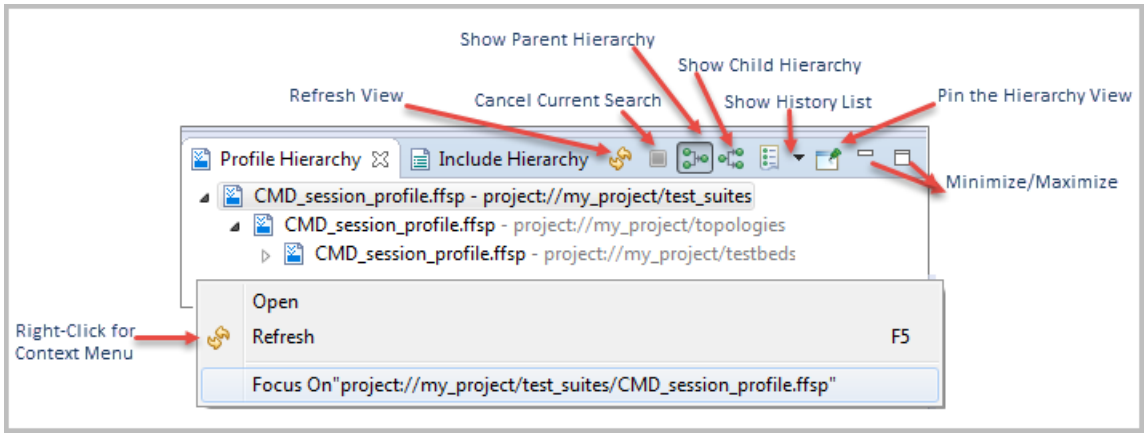
c In the **Inherits from** box, specify the session profile to inherit settings from.

- All session profiles that are defined as reference session profiles appear in the drop-down list.
- When you click **Browse**, the dialog box enables you to select from both reference and non-reference session profiles, either in the **Workspace** (the document is in the current iTest workspace) or **File system** (the document is in an itar file somewhere in the file system).

d Click **Show Hierarchy** to display the inheritance hierarchy of the session profile on the **Profile Hierarchy** page.

Note The **Show Hierarchy** button is enabled only after saving a Session Profile.

The **Profile Hierarchy** page displays the parent session and a list of children sessions that inherit settings from other session profiles. (That is, sessions added in Step a and Step c.)



Command	Description
Refresh	Refresh the selected elements and their direct children.
Cancel Current Search	Cancels the current search (useful for long running searches).
Show Parent Hierarchy	Displays all parents of the selected element.
Show Child Hierarchy	Shows all inherited session profiles used by the currently selected Session Profile.
Show History List	Displays a history of previously displayed hierarchies.
Pin the Hierarchy View	Pins the current view and allows you to open multiple hierarchy views at the same time.

Command	Description
Righ-Click	Opens a context menu with these options:
	Open: Open selected element in the default editor (Session Profile Editor)
	Refresh: Refresh the selected elements and their direct children.
	Focus On: Focus Hierarchy View on the selected element.

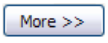
2 **Session Properties:** Specify a **Session name** for this particular set of configuration settings (the session profile). We specified **SNMP_myDUT** in this example. The name is important because:

- The name appears in the list whenever you are selecting a session to start. If you do not specify a name, then a default name like **s1** is used.
- The name appears in captured steps, making it easy to associate captured steps with a particular device.
- The name is used as the default session name when you create a test case from captured steps or add an **open** step to a test case. The session name appears in the **Session** column for each step in the session.

3 In this **SNMP example**, you will next specify the **IP address** or **hostname** for the device that you're connecting to. Other session types have different required settings. The property settings associated with each session type are described in the chapter for the particular session type in a section titled "Session profile property settings for <session type> sessions".

- Required property settings are marked with the * character.
- A blue text box indicates that the setting is being inherited from the session profile that the current session profile is based upon. If you are creating a session profile "from scratch", then the settings are inherited from the iTest default session profile. See "Property values: Inheriting settings" on page 62.
- The F indicates that you can use field replacements in the text of the property setting. See "Field replacements: Substituting values into properties and commands" on page 631.

CAUTION By default, iTest auto-validates property values as you set them. The validation process adds a marker to the property setting when there is a problem with a setting. Hold the cursor over the marker to read the details. If, instead, you configure iTest to perform validation only when you request it, then settings are not validated and no markers appear for invalid property settings. See "Controlling validation of steps and property settings" on page 169.

4 You have the option to specify additional session properties like screen color, timeouts, non-standard prompts, and so on. Click 

to open the **Session Properties** page. you will find details on property settings for each session type in the appropriate chapter.

5 **Optional. Language:** Select the language that will be used to create the session profile.

Click the **Settings** tab. On the **Settings** page, use the default language displayed (as set in "Preferences: Spirent > General > General preference settings" on page 863, Chapter 41, "Configuring iTest Preferences") or select a different language from the list.

When you select Language as Python, you may export the entire iTest test case (FFTC) to a Python script.

6 Click **Save** to save the current property settings as a session profile. We recommend a **<session type>_<Session name>.ffsp** filename convention, for example, **telnet_Myrouter.ffsp** or **snmp_10.235.34.5.ffsp**. You have the option to save the profile to the Favorites view for easy access (recommended if you expect to use the profile often to start interactive sessions — you just double-click the profile name to start a session).

Click **Start** to start the session.

Click **Reset** to reset all properties to their default settings.

You can rename and edit profiles to meet particular needs. Every time you modify session profile settings, you must save the session profile before starting the session.

Tips

- Whenever possible in test case **open** steps and device definitions, use a reference session profile that uses parameters for device- or session-specific values. For example, use a **[param ipAddress]** field replacement for the **IP address** property. This will save you a lot of time and frustration when compared with creating and maintaining a unique profile for each device (especially if the session returns a variety of prompts). See Chapter 33, "Parameters" and "Property values: Inheriting settings" on page 62.
- If you use a profile often (for example, a Telnet session with Router 5 and green text on a black background named **telnet_router5-green**), then add it to the Favorites view when you save the profile. From then on, in the Favorites view, you can just double-click **telnet_router5-green** to open a Telnet session with the settings that you prefer.
- You can launch a group of related session profiles at the same time. In the iTest Explorer or **Favorites** view, select the sessions, right-click the selection, and then select **Start All Sessions**. In the iTest Explorer, you can also right-click a folder containing multiple sessions.
- In the Test Case editor for an **open** step, you can override any session profile setting. You change the property settings in the **<Session Type> Session Properties** group (for example, **Telnet Session Properties**). See "Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step" on page 183.

Defining a reference session profile

To specify that a session profile is a reference session profile, you configure the session profile in the normal way (as described in "Defining a session profile (configuring the session settings)" on page 73) and then specify one additional property setting:

- On the **Start** page, check the **This is a reference session profile** check box.

From now on, when you are defining other session profiles, the reference session profile will appear in the drop-down **Inherit from** list on the Session Profile editor's **Start** page.

Session Profiles: Quick instructions

Note You can set preferences for the Session Profile editor. See "Setting preferences for the Session Profile editor" on page 85.

To:	Do this:
Specify that a session profile is a reference session profile	Configure the session profile in the normal way (as described in "Defining a session profile (configuring the session settings)" on page 73) and then specify one additional property setting: On the **Start** page, check **This is a reference session profile** .
Add a new session profile	1. Click  . 2. iTest opens a new profile on the **Start** page of the Session Profile editor to enable you to configure session properties (IP address, prompt definitions, and so on). Details appear in "Defining a session profile (configuring the session settings)" on page 73. When you have specified the properties, click **Save**. 3. On the **New Session Profile** dialog box, specify the following properties: **Container**: Path to the new session profile file. Default: **/my_project/session_profiles** **File name**: Name of the session profile document that you are creating (**.ffsp** filename extension). 4. Click **OK**. iTest saves the session profile and then returns to the Session Profile editor. Click **Start** to start a session.
Base a new profile on an existing profile	1. On the Session Profile editor **Start** page for the new session profile, check the **This session profile inherits settings from another session profile** check box. 2. Select the profile to use as a basis for the new profile (typically a reference profile). 3. Make any changes to property settings as needed. Click **Save**.
Edit a session profile	See "Editing session profiles" on page 80
Select Language used to create the session profile	Select the language that will be used to create the session profile. Click the **Settings** tab. On the **Settings** page, use the default language displayed (as set in "Preferences: Spirent > General > General preference settings" on page 863, Chapter 41, "Configuring iTest Preferences") or select a different language from the list. When you select Language as Python, you may export the entire iTest test case (FFTC) to a Python script.
Copy / Paste profile	In the iTest Explorer or Favorites view, right-click the session profile and select **Copy**. Paste into the appropriate folder.
Delete profile	In the iTest Explorer or **Favorites** view, right-click the session profile and select **Delete**.
Move profile	In the iTest Explorer, select the session profile and then drop it into the new folder. Alternatively: In the iTest Explorer, right-click the session profile and select **Move**. In the **Folder Selection** page, specify the new folder.
Rename profile	In the iTest Explorer or **Favorites** view, right-click the session profile and select **Rename**.

Using the ‘Update Session Profile / Testbed Device’ wizard

While working in a session, you might encounter a prompt with a format that you have never seen before. You will probably recognize that the text is a prompt and continue with your work.

When iTest executes the session, however, it may have trouble recognizing the text as a prompt because the text does not match any of the prompt definitions in the session profile. Execution might be disrupted or the text might incorrectly be interpreted as part of the command. Similar issues arise with command completion characters and command break characters.

To avoid issues like this, iTest monitors captured responses to determine whether any new prompts or special characters appeared. If so, then after the session ends, iTest starts the **Update Session Profile / Testbed Device** wizard to enable you to add the prompts and characters as configured properties in session profiles of your choice. As a result, future executions run correctly.

For an overview on how iTest recognizes prompts, see “Overview: Prompts in iTest” on page 463.

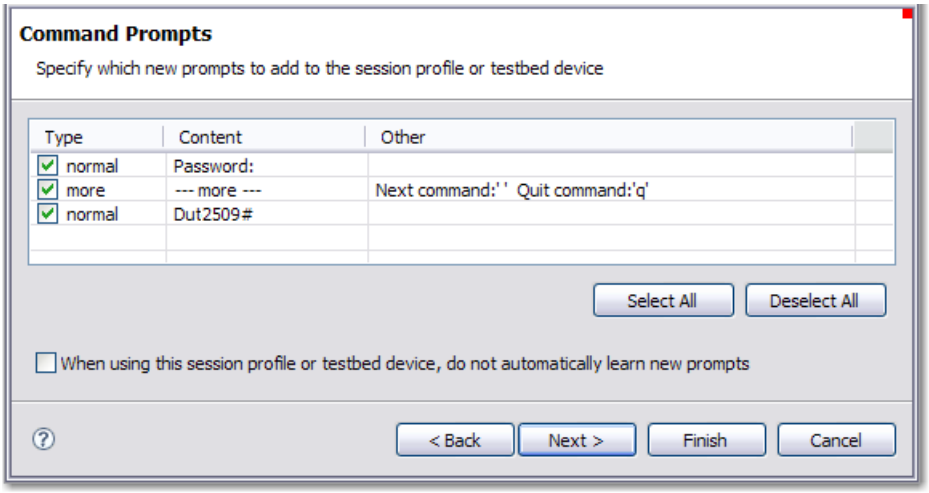
Step 1 Specify the session profiles, testbeds, or topologies to update

On the **Session Profile or Topology** page, specify the session profiles, testbeds, or topologies to update with the new property settings and then click **Next**. You can update a session defined on a testbed or topology and can also update:

- The session profile associated with the session that just ended
- If the session profile associated with the session that just ended inherits settings for another profile, then you can update any or all of the profiles in the inheritance chain

Step 2 Specify which of the text strings are actually prompts

1 On the wizard’s **Command Prompts** page, iTest lists the possible prompts and you select the ones that actually are prompts.



- The **Type** value describes the kind of prompt: **password**, **normal** or **more**.
- The **Content** value is the text that iTest noticed on the command line.

- The text in the **Other** column is the “next page” and “quit showing pages” text for **more** prompts.

Later, when you have finished running the wizard, you can edit these and other properties to customize the prompt definition. For instructions, see “Editing prompt definitions” on page 467.

2 Optional: The **When using this session profile or topology device, do not automatically learn new prompts** option tells iTest that, for future test runs, do not start the wizard to present any prospective prompts. Instead, treat all text as a response and use the Completion settings to determine when the step is finished executing.

Step 3 Identify new command completion characters

When you perform manual testing, you frequently use characters like tab to auto-complete partially typed commands. To enable iTest to capture the completed form of commands and discard the incomplete form, it must know all command completion characters that you might use while executing test cases.

The wizard’s **Command Completion Characters** page is populated when iTest has noticed command completion characters that have not yet been configured.

Step 4 Identify new command break characters

Most devices interpret certain characters to mean “break execution”. For example, Ctrl-C is a commonly used break character.

The wizard’s **Command Break Characters** page is populated when iTest has noticed break characters that have not yet been configured.

Step 5 Finish

The **Finish** page gives you an opportunity to review what you have done before committing the changes.

Editing session profiles

Use the Session Profile editor to make changes to any session profile properties.

1 In the iTest Explorer or **Favorites** view, right-click the profile and select  **Open**.

2 The profile opens in the Session Profile editor, as described in “Defining a session profile (configuring the session settings)” on page 73.

CAUTION When the editor opens, it displays the session type for the session profile that you are editing. If you change the **Session type** property, then ***all*** property settings are replaced with the default settings for the newly selected session type.

3 Move to the appropriate editor page and modify the property settings as needed.

Session Profile editor: Start a New Session page See page 81.

Session Profile editor: Settings page See page 82.

Session Profile editor: Global Events page See page 84.

Session Profile editor: Global Rules page See page 84.

Session Profile editor: Parameters page See page 84.

Session Profile editor: Response Filters page See page 85.

Session Profile editor: Start a New Session page

On the **Start** page of the Session Profile editor, you can:

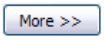
- Specify session property settings and then save the settings to a session profile document
- Define a new session profile based on an existing session profile
- Modify an existing session profile

When you start a newly defined session, iTest asks you to name the session profile document and asks whether to save it in the Favorites view (so that you can later double-click the file to start a session with the identical properties).

Tip You can set preferences for the **Start** page. See "Setting preferences for the Session Profile editor" on page 85.

New sessions start in an appropriate session window.

More button

Click 

to view the property settings that you configure when you launch a new session using the Start Session dialog. The navigator view enables you to select the property group to modify.

- To save your changes to the session profile document, click **Save**.
- To start the session as defined, click **Start** (you'll be asked to save the session profile first).
- To reset all property settings to default, click **Reset**.

Advanced Users Many of the property settings for session profiles support field replacements to enable you to parameterize settings so they can be determined dynamically before an automated test case starts the session. You might use **param**, **profile**, **scriptEval**, and **tcl** command field replacements to improve the flexibility and portability of automated test cases. Sometimes, to perform an interactive test, you might need to manually start a session that typically starts only for automated test sessions. To enable you to do this, if iTest encounters any **param**, **profile**, **scriptEval**, or **tcl** command field replacements while starting a session, iTest starts a Tcl interpreter so that the field replacement can be resolved before the session starts. When the session ends, the Tcl interpreter is disposed. When the session is restarted by pressing Enter, the substitutions are not made again. If a Tcl interpreter service is requested on restart, however, a new interpreter will be created and returned.

Session Profile editor: Settings page

The settings on the **Miscellaneous** page apply whenever the session profile is used in a test case.

General Information

Headline	Supply a brief description to help co-workers understand how the session profile should be used. The text that you type here appears in the **Headline** column of the Favorites view to help you when selecting a session profile.
Description	Optional. Type additional documentation to help coworkers to understand and maintain the session profile
Language	Use the default language displayed (as set in "Preferences: Spirent > General > General preference settings" on page 863, Chapter 41, "Configuring iTest Preferences") or select a different language from the list. When you select Language as Python, you may export the entire iTest test case (FFTC) to a Python script.

QuickCalls

QuickCall Library	When you specify a QuickCall library here, any test case step in the session searches the QuickCall library for the requested QuickCall. In addition, all QuickCalls in the library are listed when you request a QuickCall during a manual session. See Chapter 9, "QuickCalls: Defining and using a library of custom actions"

Response Map Library

Response Map Library	Optional. The response map library that you specify here is searched by any test case step in the session when the step attempts to determine which response map to apply to a response. **Note** When a session profile (ffsp file) is saved, if Response Map Library is blank, iTest automatically inserts the project address where the session profile is being saved. See Chapter 26, "Response Maps: Returning Data from Responses" **Tip** The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries. See "Making use of existing response map libraries: Chaining response maps" on page 559.
External source	Optional. Specify the response map or response map library that contains the sample response that a step should use to emulate the response. The use of this setting is fully described in "Step 3: Activate emulation for particular steps or sessions and specify the source of the emulated response" on page 478. By default, the source that you specify here is inherited by the **External source** property for the step. If a source is specified in the **External source** property for the step, the value in the step properties override the setting that you specify here, To enable the test case to dynamically determine the sample response at runtime, field replacements are supported in this field. **Note** Often, the **External source** might be the same as the **Response Map Library** for the session profile. The two properties are distinct because, during emulation, you might want to use different responses than you use during actual execution with response mapping (including error cases). Default: [empty] **Tips** The response map chaining feature enables you to specify that, during the search for the sample response to use for emulation, any step that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries. See "Making use of existing response map libraries: Chaining response maps" on page 559. On the Response view, clicking the **Add This Sample to an Existing Response Map**  is a quick way to add a sample response for use in an emulated step. See "Response view" on page 291 and "Creating a response map: Instructions" on page 554.

Form Map Library

Form Map Library	The form map library that you specify here is searched by any test case step in the session when identifying the targets on a page. **Note** When a session profile (ffsp file) is saved, if **Form Map Library** is blank, iTest automatically inserts the project address where the session profile is being saved. See Chapter 40, "Form Maps"

Emulation Source

Emulation source	Specify the Response map library that should provide the responses for sessions. Specify either a library project or the catalog for a library. See Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions"

Session Profile editor: Global Events page

On the Events page, you specify the actions that should take place whenever particular events occur during execution. Events are described more fully in "Events" on page 603.

Actions that you define here apply to any step in a session that is based on the current session profile. See "Configuring Events: The Global Events page" on page 603

Session Profile editor: Global Rules page

On the Global Rules page, you define analysis rules that apply to all steps that are in a session based on the current session profile. See "Global Analysis Rules page" on page 665.

Session Profile editor: Parameters page

Parameters that you define for a session profile can be accessed by any step in a test case that operates in a session that was started using the session profile.

The **Parameters** page appears in several editors. For details on creating and managing parameters, see "Masking a parameter's value" on page 744.

Important note about field replacements in session profile property settings

While you can specify field replacements in many session profile property settings, substitution occurs for some commands only at runtime (because a value required by the command is obtained from the execution context).

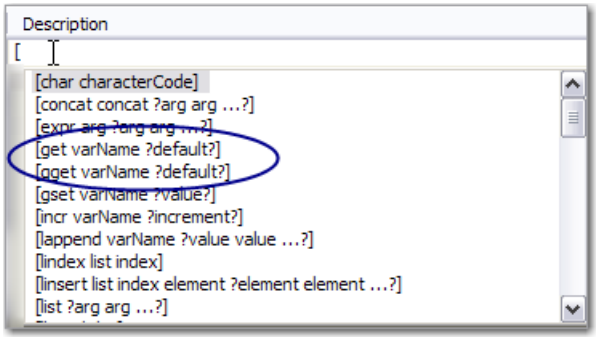
For example, the **param** command is replaced at runtime because it obtains the parameter value from the heap. The **char** command, in contrast, can be replaced at any time, because it does not rely on the execution context.

As a result, when you start a manual (interactive) session, **char** commands are replaced, but **param** commands are not. The session cannot start.

You can avoid this issue by specifying the optional **default** argument for any command that is replaced at runtime. The command is replaced during manual sessions using the default value that you specify for the **default** argument.

To determine whether a command are replaced only during execution (and therefore requires a value for the **default** argument to enable you to start manual sessions), type the **[** character into the **Description** cell for the step — iTest displays the list of commands and their syntax. Any

command that uses an optional **?default?** argument is replaced during execution. In this example, you can see that the **get** and **gget** commands allow you to specify a default value.



Session Profile editor: Custom Types

The **Custom type** tab allows you to define custom parameter type with a set of named value elements. The custom parameter type and elements you define displays as an option that you can select to indicate the parameter types and values.

See "Defining Custom Types" on page 750.

Session Profile editor: Response Filters page

You can use ***response filters*** to remove unwanted text from a response after a step has executed and before iTest applies analysis rules. The "filtered" response (the portion of the response that remains) is typically cleaner to read and easier to map and to understand. As a result, the sample responses that you use to create Response maps are simpler, and test reports are more readable.

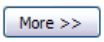
See "Defining response filters" on page 543.

Masking passwords during capture and replay

You will typically want to replace password text with * characters to mask them from unauthorized use. By default, iTest replaces commands that the device does not echo with eight * characters (passwords are typically not echoed).

The default setting causes the following behavior: Before creating a Capture report, iTest masks all command text for which no echo was returned.

Masking passwords

1. While editing the session profile, click 

to view the property pages.
2. Click **Terminal > Capture**.
3. On the **Capture** page, check **Mask unechoed commands**.

Setting preferences for the Session Profile editor

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Editors > Session Profile Editor**.

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Spirent > Editors > Session Profile Editor

Display a blank New Session page after starting a session	Check the box to open the Session Profile editor whenever a session starts. This is useful when you need to start an additional manual session while the current session is active. The alternative is to click **Start a New Session** whenever you need to work on session profiles. This preference setting s then set to unchecked. Default: checked
Display a blank New Session page after switching to the Manual Testing perspective	Check the box to open the Session Profile editor whenever you switch to the **Manual Testing** perspective This is useful when you are in the process of designing new session profiles. The alternative is to click **Start a New Session** whenever you need to work on session profiles. Default: unchecked
Show the introduction page for the Update Session Profile wizard	This option enables you to return the page for display if you had previously selected the **Do not show me this page again** option in the wizard. Default: checked

CHAPTER 5

Testbeds

iTest Testbeds

You use the Testbed editor to define a ***testbed*** document, that is, to specify all of the physical devices (or sessions with software applications) that make up a particular testbed.

A testbed document is a collection of the information that test cases need to access the devices and sessions (for example, the IP address of the device and login information). Each named device listed in the testbed is based on a default iTest session type (such as Telnet, Web, SNMP, and so on) or on a session profile.

When you associate a testbed with a test case, then all of the named devices are available to start sessions in the test case. To run a test case on a different set of physical devices (that is, a different testbed), you update the test case to refer to a different testbed document.

You will probably not spend too much time in the Testbed editor — it's primarily a configuration tool.

Important Before you define a testbed, check whether it might be better to define a topology instead. See Chapter 24, "iTest Topology editor".

Why it is a good idea to define testbeds

Testbeds are powerful in many ways:

- Once you have defined the testbed, a test case developer can open the **General** page on the Test Case editor and specify the **testbed** for the test case. As a result, when they add an **open** step to the test case, the list of available devices that they see is the list that you specified for the testbed — no guesswork on the test developer's part.
- When you define a device by basing it on a default session type, you avoid cluttering your system with many session profile documents that might differ only in small details. Easier to maintain and to understand.
- You can design your test cases to be easily run against a variety of testbeds (for example, with devices that differ only in software version, and so on). As a result, you can run a test against several different sets of physical devices by changing a single setting; the testbed for the test case.

Reference testbeds

If you expect to define several testbed documents that represent similar device configurations, then you will benefit by defining a ***reference*** testbed to act as a template.

You use a reference testbed to store the common configuration settings for a similar group of devices. When you base testbed **B**, for example, on reference testbed **A**, then testbed **B** inherits all of its property settings from testbed **A**. You can then make minor changes to testbed **B** to make it ready to use for a different set of devices.

For example, several testbeds involve one RX5000 router, one traffic generator, and one other device. Between each testbed, the group of devices share most settings but each has a unique IP address. You create a single reference testbed **A** that you can use as a template for each other testbed. In the other testbeds, you set the **Inherit from** property to testbed **A**. You then specify appropriate **IP address** property settings for each device and then save the file as testbed **B**.

Specifying a Global testbed

If you configure a particular testbed as the ***Global testbed***, then, when you execute any test case that does not specify a Local testbed, iTest asks whether you would like to use the Global testbed for execution. You can configure the behavior so that particular test cases ignore the Global testbed while others use it. See “Global testbed” on page 94.

Specifying testbeds when executing iTestRT or itestcli (headless execution)

When you execute a test case using a headless version of iTest, you can use a flag to pass in the testbed to use. The specified testbed overrides the **Local testbed** that is configured in the test case and the Global testbed.

Designing test cases to be portable to several testbeds

1. On the **General** page of the Test Case editor, specify the **testbed** for the test case, (for example, **personal_testbed**).
2. Use **device:device_name** URIs in the **open** steps (the full list of devices appears as a drop-down list in the **Description** cell).
3. To use a different set of devices, edit the test case to specify a different testbed (for example, **regression_testbed**). The **regression_testbed** document refers to a different set of devices. For example, the device **Names** in the **regression_testbed** testbed must be identical as in **personal_testbed**, but might have different IP addresses or other configuration settings.

Options for improving portability

The following options and several variants are supported by iTest:

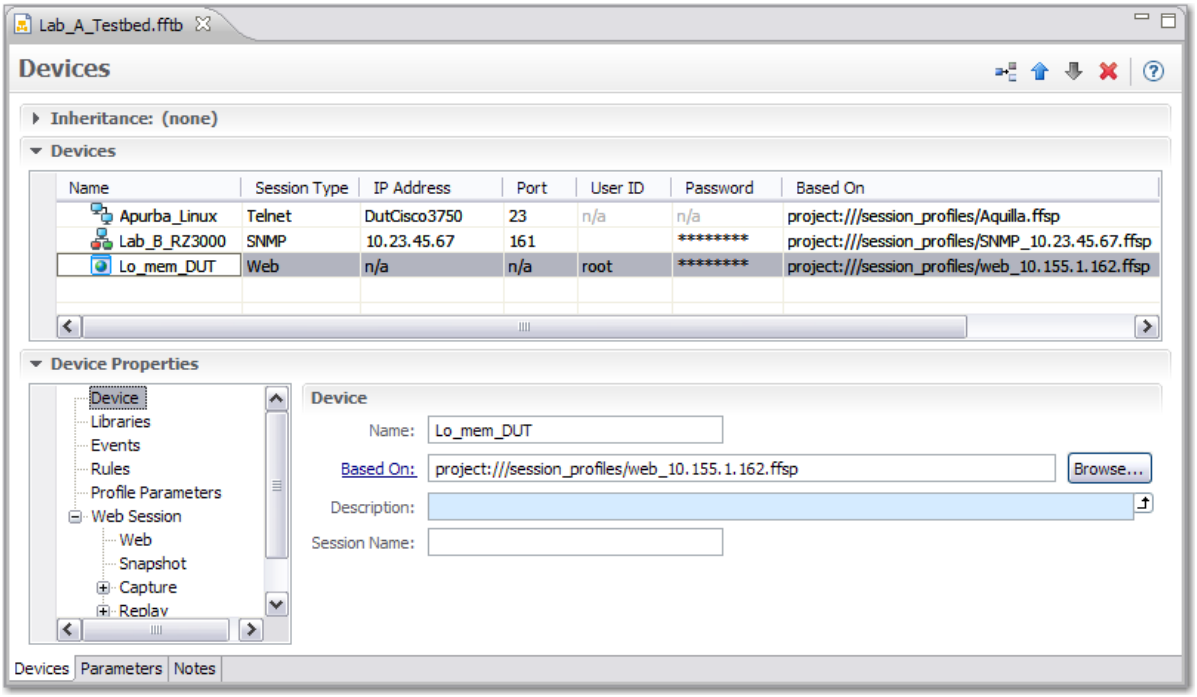
- Pass parameters in to the test case. If you parameterize information that identifies a device (for example, IP address, hostname, and port) in session profiles or **open** steps, you can avoid having to define testbeds entirely.
- If you use dynamically allocated testbeds, your proprietary automated regression system might create new XML session profile documents on the fly.
- You might store statically defined session profiles for each device in your testbed pool. At runtime, your proprietary software can then dynamically generate an XML testbed file that specifies the subset of devices to use in the test.

Overview: Creating a testbed

Defining a testbed consists of creating the list of devices that a test case will open sessions with. In addition, you can specify parameter values for the testbed. A testbed document includes one session profile (reference, custom, or default) for each device.

Tip Whenever the Testbed editor is selected, the **Testbed** menu appears in the menu bar. The options that appear in the menu vary depending on the editor page in use.

In the **Devices** grid, each line represents one device. The properties for each device appear in the Device Properties section.



Once you have defined the testbed, a test case developer can open the **General** page on the Test Case editor and specify the ***Local*** testbed for the test case. As a result,

- For **open** steps in the test case, the test case developer can easily select any of the devices defined in the testbed
- The parameters that you define for the testbed can be accessed by any step in a session that is based on a session profile specified in the testbed.

There is another option for specifying the testbed that a test case should use; the Global testbed.

Adding and configuring a testbed

The pages of the Testbed editor enable you to configure various collections of testbed properties. The **Devices** page is the most important; it lists the devices (or other session types) that will be available to executing tests. To create a testbed:

1 Click **New > Other**.

2 On the **New** page, select **iTest > Testbed**. Click **Finish**.

3 On the **New Testbed** page, specify the path (**Container**) and **File name** for the file. Click **Finish**. The file is created and opened in the Testbed editor.

4 On the Testbed editor's **General** page, document the testbed for the benefit of coworkers, as described in "Testbed editor: General page" on page 93.

5 On the editor's **Devices** page, open the **Inheritance** section (click ▸). Select one of the following options:

- **This is a standalone testbed**. A standalone testbed does not inherit device definitions from any other testbed. For example, you might define a standalone testbed and then identify it as a reference testbed in a subsequent step.
- **This testbed inherits settings from another testbed**. In the **Inherits from** box, specify the testbed to inherit settings from. iTest then populates the **Devices** grid with all of the devices in the specified testbed. If you base the testbed on a reference testbed, then you may not need to take many of the steps remaining in this procedure.
  - All testbeds that are defined as reference testbeds appear in the **Inherits from** drop-down list.
  - When you click **Browse**, the dialog box enables you to select from both reference and non-reference testbeds. You can browse for a testbed either in the **Workspace** (the document is in the current iTest workspace) or **File system** (the document is in an itar file somewhere in the file system).

6 Each line in the **Devices** section defines a device. Add a device by clicking  .

7 In the **Name** cell, provide a meaningful name for the device. This is the text that appears in the **Select a Session Profile or Device** dialog box when a test case developer is selecting a device for an **open** step.

The Name is very important. Here is an example that shows why: In this example, we name a device **DUTRouter2** in both the **LocalTestbed** testbed (the document that defines your personal development testbed) and the **SharedRegressionTestbed** testbed (that defines the QA regression testbed) — only the IP **addresses** are different.

As a result, any test case that refers to **DUTRouter2** in a step can, without modification, use either of the two testbeds (you have only to specify the appropriate testbed on the Test Case editor **General** page or use the Global testbed). Once you finish developing the test case, you can hand it off for regression without any changes.

---

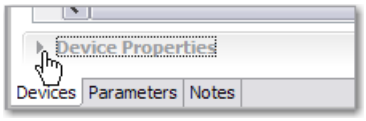
8 You can edit the other cells as noted in the table. Any changes you make to properties for a device (either in the **Devices** grid or in the property pages) overwrite the settings in either the session profile or default session type specified in the **Inherits from** cell.

CAUTION By default, iTest auto-validates property values as you set them. The validation process adds a marker to the property setting when there is a problem with a setting. Hold the cursor over the marker to read the details. If, instead, you configure iTest to perform validation only when you request it, then settings are not validated and no markers appear for invalid property settings. See "Properties in: Spirent > Editors > Test Case Editor" on page 190.

CAUTION Inheritance is turned off for a property if you change the value of the property. To revert to the inherited value and to turn inheritance on, you must either undo (Ctrl+Z), or change the inheritance setting in the **Device Properties** pages.

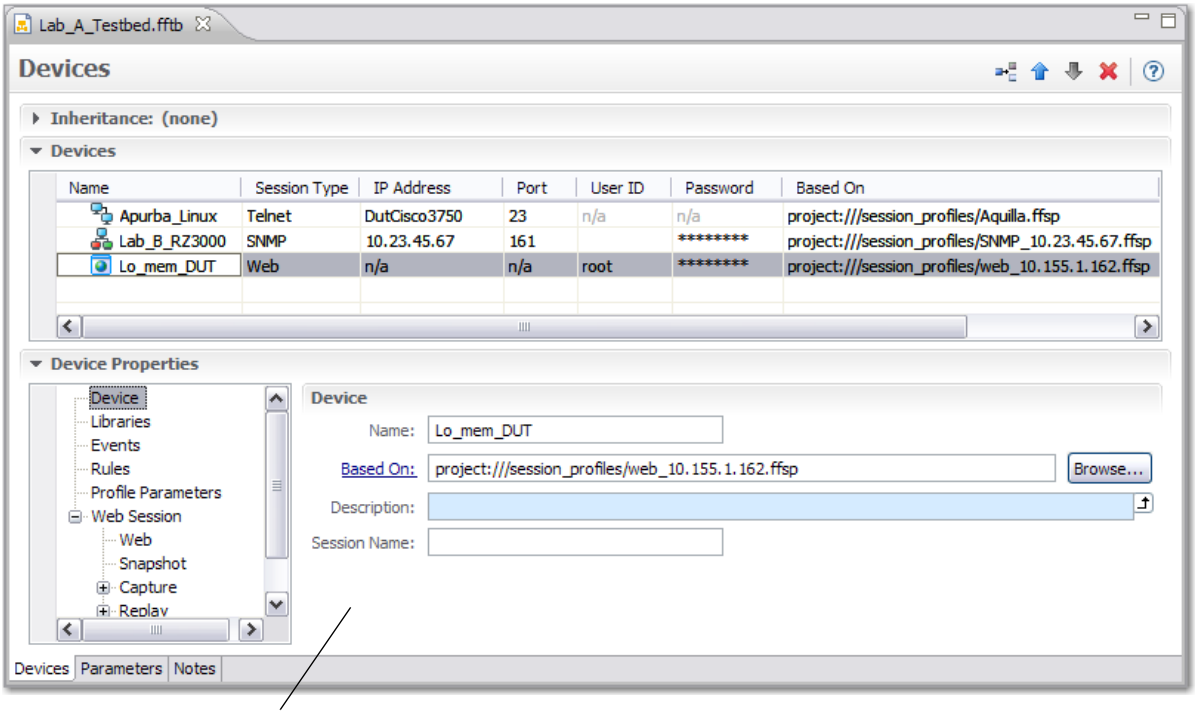
Name	Provide a meaningful name for the device (for example, Z500Router_12 or Room6_ Z500Router). This is the text that appears: In the Select a Session Profile or Device dialog box when a test case developer is selecting a device for an open step. As the URI in the test case Description cell and in other referring URIs (in the form device:Z500Router_12) You cannot edit this value if the testbed is based on a reference testbed.
IP Address	Hostname or IP address of the device. Field replacements are supported.
IP Port	IP port number used to connect to the device. Field replacements are supported.
User ID / Password	Authentication credentials. Many session types do not use credentials. Passwords are masked. Field replacements are supported.
Inherits from	URI identifying either: The default session type (application) that the device is based on. For example: application://com.fnfr.svt.applications.snmp or The session profile that the device is based on (typically a reference session profile). All reference session profiles are listed in the drop-down list. This column does not appear if the testbed is based on a reference testbed. **Tip**: If the session profile includes parameter definitions, then, during execution, the **Session profile** name can appear in the Data view and you can make changes to parameter values.
Session Type	Session type of the session profile that appears in the **Inherits from** cell. You cannot edit this value.

9 If appropriate, edit the property settings for the device. Click the arrow next to **Device Properties** to open the section.



Any changes that you make in the **Device Properties** section of the Testbed editor overwrite the settings in the default session type profile or session profile.

As with values in the grid, inheritance is turned off for a property if you change the value of the property.



This set of properties for the selected device is exactly the same as the set of properties that you defined in the session profile that is specified for the device. Session profile properties are discussed in “Session Profile editor: Start a New Session page” on page 81.

During execution, the changes that you make here override the settings in the session profile that is specified for the device.

10 Repeat the process for each device in the testbed.

11 Optional. If the testbed that you are configuring is a reference testbed (as described in “iTest Testbeds” on page 87): On the **General** page, check the **This is a reference testbed** check box.

From now on, when you are defining other testbeds, the reference testbed will appear in the drop-down **Inherit from** list on the Testbed editor’s **Devices** page. In addition, the testbed will appear when you click **Browse** to specify the testbed to inherit from.

12 Save the testbed document.

Testbed editor: General page

Use the **General** page to document the testbed for the benefit of coworkers.

Headline	Optional. Type a one-line description of the testbed. For example, **LabA_Switches** or **newRoutersWithTrafficGen** This text appears in the Favorites view for the testbed.
Description	Optional. Type text that describes the testbed to make its usage clear to coworkers.
This is a reference testbed	Check the box to specify that this is a reference testbed, as described in “iTest Testbeds” on page 87.

Testbed editor: Devices page

A iTest testbed is made up of devices. A device is the set of configuration settings that lets iTest open a session with a device or with an application (like a Swing application).

You use the **Devices** page to specify the devices that make up the testbed.

You can specify the devices either by adding them individually or by referring to another testbed and “pulling in” all of its devices:

- **Standalone testbed**: You add device definitions in the **Devices** section. Each device is based on a session profile or a default session type. You can modify many of the properties of the devices.
- **Inherits from** testbed: (that is, inherit the property settings of this testbed on an existing testbed). You have the option to modify many of the properties of the devices.

Devices page context (right-click) menu

These tools (and their counterparts in the toolbar) are disabled when the testbed references another testbed. Right-click on the Testbed and choose the required action.,

Migrate to Topology	Click **Mirgrate to Topology** to mirgrate the testbed to a corresponding topology.
Insert Device	Add a device to the end of the list or after the selected device.
Move Selected Devices Up / Down	Move the selected devices up or down in the list. The list of available devices for an **open** step in the test case editor follows this order.
Delete	Delete the selected devices from the list.
Open	Edit the selected device (in the Session Profile editor.
Cut / Copy / Paste / Select All	Typical editing tools.

Devices page toolbar

These tools (and their counterparts in the right-click menu) are disabled when the testbed references another testbed.

	Add a device to the end of the list or after the selected device.
	Move the selected devices up or down in the list. The list of available devices for an **open** step in the test case editor follows this order.
	Delete the selected devices from the list.

Defining a reference testbed

To specify that a testbed is a reference testbed, you configure the testbed in the normal way (as described in “Adding and configuring a testbed” on page 89) and then specify one additional property setting:

- On the **General** page, check the **This is a reference testbed** check box.

From now on, the testbed will appear in the drop-down **Inherit from** list on the Testbed editor’s **Devices** page. In addition, the testbed will appear when you click **Browse** to specify the testbed to inherit from.

Testbed editor: Parameters page

Once you have defined the testbed, an automation engineer can open the **General** page on the Test Case editor and specify the **Local topology or testbed** for the test case. As a result, the parameters that you define here can be accessed by any step in a session that is based on a device or session profile in the testbed.

See “Working with parameters: The Parameters page” on page 733.

Global testbed

If you identify a particular testbed as the ***Global*** testbed, then, when you execute any test case that does not specify a Local testbed, iTest asks whether you would like to use the Global testbed for execution. You can configure the behavior so that particular test cases ignore the Global testbed while others use it. (For more information on Local testbeds, see “Test Case editor: General page” on page 162.)

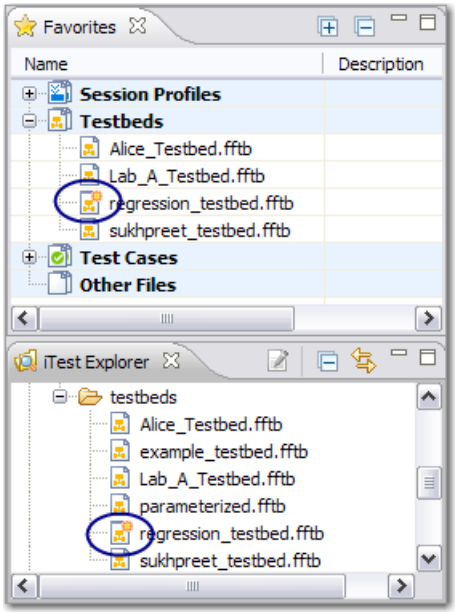
The testbed that was used for execution is identified in the Execution view, in the Step Issues view, and in test reports.

Specifying a testbed as the Global testbed

In the iTest Explorer or Favorites view, right-click the testbed and select **Set As Global Testbed**.

To undo the setting, right-click the Global testbed and select **Clear Global Testbed** or specify that another testbed is the Global testbed.

The icon for the Global testbed is marked to distinguish it from other testbeds:



Testbed used during execution

You have the option to modify the default action noted here by using the **Preferences** page to configure iTest to use either testbed. See “Setting preferences for execution” on page 281.

Global testbed specified?	Local testbed specified?	Testbed used during execution
No	No	None
Yes	No	The default action is to display a dialog box that asks for your choice.
Yes	Yes	The default action is to display a dialog box that asks for your choice.
No	Yes	Local

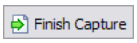
CHAPTER 6

Capturing Manual (Interactive) Sessions

Overview: Creating a test case by capturing interactive sessions

You can create a test case by typing steps into steps grid of the Test Case editor, but the easiest way to create a test case is one of the following methods:

- Use the **Develop a test case** page on the **iTest Activities** perspective to manually perform sessions and specify which captured steps to add to the test case. See "Developing a test case" on page 17.
- **Direct-to-test capture.** In the **iTest Development** perspective, click 

, perform the steps interactively (manually), and then click 

. iTest converts the captured steps into an automated test case and displays it in the Test Case editor. For details, see "Saving interactive steps as steps in a test case: The 'Add to iTest Test Case' wizard" on page 98.
- **Save a captured session as a procedure.** This method allows you to select any set of captured steps in the Capture view (regardless of when they occurred) and add them to a test case. See "Adding captured steps into a test case or Python Script" on page 100.

In any case, the **Add to ITest Test Case** wizard starts and gives you the option to either:

- Create a new test case and add the steps to it
- Add the steps to an existing test case

Note For generating Python Scripts from the captured steps, see Chapter 59, "Python Automation Library", section "Capturing Manual (Interactive) Sessions" on page 97.

Capturing in the debug mode

iTest also supports capturing of actions performed in interactive session during debug mode. These steps will be added to Capture View and will be **green** in color (by default). Currently, the following sessions captures are supported in debug mode:

Chat, Command, Database, File, Flex, Http, Mail, Process, REST, SNMP, Serial, SSH, Swing, Syslog, Tcl Shell, Telnet, UDP, Web, Web Service, Wireshark, XML-RPC, VNC

You can set your preferences for capturing actions required in debug mode. See "Setting preferences for the Capture view" on page 111.

Other options for creating a test case

See "Other options for creating a test case" on page 125.

Saving interactive steps as steps in a test case: The ‘Add to iTest Test Case’ wizard

Note For generating Python Scripts from the captured steps, see Chapter 59, “Python Automation Library”, section “Capturing Manual (Interactive) Sessions” on page 97.

When you click Start Capture, you tell iTest to insert into a test case any actions that you take in sessions from now until you click the Finish Capture.

This direct-to-test method of placing your manual steps into a test case is the fastest way to create (or to add steps to) test cases. You have the option of inserting individual steps into an existing test case, inserting whole sessions into an existing test case, or of creating a new test case.

- When you select a step in a test case before you start direct-to-test capture, then the wizard adds the steps after the selected step.
- When you start direct-to-test capture without having selected a step, the wizard gives you the option to save the captured steps as a procedure in an existing test case or as a procedure in a new test case.

How direct-to-test capture works

- If you start direct-to-test capture while a session is open, then the wizard uses **Step mode**. As a result, only steps in that session are added to the test case. Actions in other sessions are not direct captured (iTest will remind you whenever you perform an action in another session). For this reason, if you want to direct-capture multiple sessions, then close all sessions, click Start Capture, and then start the manual sessions.
- If you first select a step, click Start Capture, and then start a new manual session, then, when you click Finish Capture, the entire manual session is added after the selected step as a new procedure.
- If you select a step that is not associated with a session before you start direct-to-test capture, then the wizard uses **Session mode**. As a result, all steps in the interactive session are added to the test case, even if you performed only a few steps after clicking Start Capture.
- Comments and markers in manual sessions are saved as **comment** steps. (During interactive sessions, you add comments using the Capture Comments view and you add markers using **Insert Marker** in the Capture view.)

For devices with more than one session attached or for multiple captures that use the same session profile: To create the Session IDs that appear in the **Session** cells in the Test Case editor and in test reports, Spirent iTest uses the combination of **Session name** from the session profile (for example, **myDUT**) and a unique session number for the day. (for example, **myDUT.1** and **myDUT.2**). If the session profile does not specify a **Session name**, then Spirent iTest uses the filename of the session profile in its place.

Using direct-to-test capture

1. Optional. In the Test Case editor, select a step. The new steps are added after the selected step.

   If you do not select a step, then the captured steps are added either to a new test case or to the end of an existing test case.

2. Click Start Capture.

3. Start sessions and perform the steps that you want to add to the test case.

4. Click Finish Capture.

5. The **Add to iTest Test Case** wizard starts.

6. On the **Test Case** page, you have the option to create a new test case and add the procedure to it or to add the procedure to an existing test case. In either case, iTest suggests a project, folder, and test case file name into which to add the new steps. In the **File name** field, type or browse to the path and name for the test case.

Create a new test case using the captured items	When you finish with the wizard, iTest creates a new test case, opens it in the Test Case editor, and then adds the procedure steps to the test case. You can then edit and save the test case as needed. If you use a test case template, then, on the next wizard page, you will specify the procedure to add the steps to. The steps are added after any existing steps in the specified procedure.
Add captured items to an existing test case	When you finish with the wizard, the Test Case editor opens to the specified test case and then iTest adds the procedure after the last test case step. The Test Case editor remains open.

7. On the **Insert** page, specify whether to add the new steps as a procedure or as individual steps after the steps that you had selected in the test case before starting direct-to-test capture.

Tip If you have used the wizard before and you feel confident that the wizard will take appropriate actions, then you can click **Finish** at any time.

8. The **Procedure** page appears only if you chose to add a procedure to an existing test case. On the **Procedure** page, specify the following values and then click **Next**.

Name	Type the name of the procedure. This is the name that test case developers will use to call the procedure. Alternatively, if you use a test case template, then, from the list, select the procedure to add the steps to. For example, **GetPortSettings** or **CleanUpTestbed**.
Headline	Optional. Type a single line of text that describes the procedure. This string will appear with the procedure name in the drop-down list of procedures in the Description cell for call steps or **CallProcedure** actions. This text also appears in the **Headline** column of the Favorites view to help you when selecting a procedure.

9 On the **Finish** page, if you are confident in your selections, click **Finish**. The steps are added to the specified test case. Move the procedure or steps and edit as needed and then save the test case.

Tip If you are confident in your selections at any point while using the wizard, you can click **Finish** to add the steps.

Adding captured steps into a test case or Python Script

Note For generating Python Scripts from the captured steps, see Chapter 59, "Python Automation Library", section "Capturing Manual (Interactive) Sessions" on page 97.

While you can add a procedure to a test case manually by typing it into the Test Case editor, the fastest way to add a procedure is to perform the steps manually, select the captured steps or sessions in the Capture view, and then use the **Add to iTest Test Case** wizard.

- If you select multiple sessions, they are saved together in capture order as a single procedure.
- Comments and makers are converted into EXEC comment actions.
- When you add a captured session to a iTest test case, each QuickCall that you performed during the manual session becomes a ***single step*** in the test case (regardless of how many actions the QuickCall actually performed). This improves readability, portability, and consistency. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

Adding captured sessions or steps into a procedure in a iTest Test Case

1 In the Capture view, select one or more captured items (use Ctrl-click and Shift-click to select multiple items).

Important If you select items from more than one session, then you must include an open step for each session

2 Click **Add to iTest Test Case**  . (Alternatively, right-click the selection and select **Add to Test Case**.) The **Add to Test Case** wizard opens.

3 On the **Test Case** page, you have the option to create a new test case and add the procedure to it or to add the procedure to an existing test case. In either case, iTest suggests a project, folder, and test case file name into which to add the procedure. In the **File name** field, type or browse to the path and name for the test case.

Create a new test case using the captured items	When you finish with the wizard, iTest creates a new test case, opens it in the Test Case editor, and then adds the procedure steps to the test case. You can then edit and save the test case as needed. If you use a test case template, then, on the next wizard page, you will specify the procedure to add the steps to. The steps are added after any existing steps in the specified procedure.
Add captured items to an existing test case	When you finish with the wizard, the Test Case editor opens to the specified test case and then iTest adds the procedure after the last test case step. The Test Case editor remains open.

4 On the **Procedure** page, specify the following values.

Tip If you have used the wizard before and you feel confident that the wizard will take appropriate actions, then you can click **Finish** at any time.

5 The **Procedure** page appears only if you chose to add a procedure to an existing test case. On the **Procedure** page, specify the following values. You will define the procedure more fully in the Test Case editor. Click **Next**.

Name	Type the name of the procedure. This is the name that test case developers will use to call the procedure. Alternatively, if you use a test case template, then, from the list, select the procedure to add the steps to. For example, **GetPortSettings** or **CleanUpTestbed**.
Headline	Optional. Type a single line of text that describes the procedure. This string will appear with the procedure name in the drop-down list of procedures in the Description cell for **call** steps or **CallProcedure** actions. This text also appears in the **Headline** column of the Favorites view to help you when selecting a procedure.

6 On the **Generate QuickCall** wizard, specify the following values.

Generate QuickCall in the associated QuickCall Library	Select to generate the QuickCall into the associated QuickCall library of the referenced session profile. Not selecting or clearing your selection prevents the QuickCall from being generated and the rest of the options wills not be available for input or selection.
Procedure Name	Type the name of the procedure. This is the name that test case developers will use to call procedure/QuickCall.
Procedure Description	Optional. Type a single line of text that describes the procedure. This string will appear with the procedure name in the drop-down list of procedures in the **Quickcall**.
Delete the open and close steps	Select to delete all the open and close steps during the generation process, and substitute the session ids with **$session**. If you do not select the option, then the session Ids are not replaced.
Create Quickcall Library if it does not exist	**Note** The **Create Quickcall Library if it does not exist** option is enabled only when there is no QuickCall library associated with the referenced session profile. Enabled when **Generate QuickCall in the associated QuickCall Library** is selected. When the option is enabled and selected, the wizard does the following: • creates a QuickCall library in the default library and QuickCall library associated with the referenced session profile • Add the captured steps to the QuickCall library. For example: **project://my_project/libraries/Command_prompt_quickCall_library.fftc.** Click **Browse**, navigate to the required target location, select location and click **OK**. Click **Finish** and iTest generates the QuickCall procedure in the specified library. **Note** An error message displays if the library extension.**fftc** is not specified.

See also Chapter 9, “QuickCalls: Defining and using a library of custom actions”

7 On the **Finish** page, click **Finish**. The Test Case editor opens, displaying the procedure at the end of the test case. Edit as needed and then save the test case.

Tip If you are confident in your selections at any point while using the wizard, you can click **Finish** to add the procedure.

Overview: Capture view

iTest records every step in any interactive session that you start. The Capture view lists every session and step that iTest has captured. In addition, you can add comments and markers to the captured steps.

You can use the information in the Capture view for a variety of purposes:

- You can select items to create a capture report that documents what has happened.

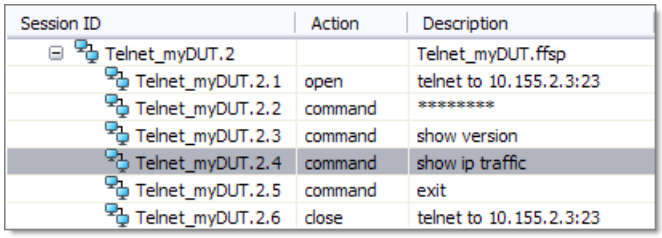
- You can drop captured steps into an existing interactive session to replay the steps.
- You can use selected steps or whole sessions to generate a new automated test case or add the steps to an existing test case.

Important For CLI sessions, iTest also captures the unique text of the prompts returned by the session. As you can imagine, the ability to distinguish between a prompt and an extended wait for a response is very important during automated execution. iTest does much of the work of learning prompts behind the scenes, and we'll discuss how you can help to train iTest to recognize prompts.

Working in the Capture view

iTest ***always*** records your interactions with sessions — capturing session **open** and **close** actions, the actions that you send to sessions on devices, each session's responses, and so on. iTest displays the ordered list of captured items in the Capture view.

Note iTest captures sessions regardless of whether the Capture view is open or not.



Session ID	Action	Description
Telnet_myDUT.2		Telnet_myDUT.ffsp
Telnet_myDUT.2.1	open	telnet to 10.155.2.3:23
Telnet_myDUT.2.2	command	********
Telnet_myDUT.2.3	command	show version
Telnet_myDUT.2.4	command	show ip traffic
Telnet_myDUT.2.5	command	exit
Telnet_myDUT.2.6	close	telnet to 10.155.2.3:23

In a Telnet session, for example, when you type a command and press Enter, iTest captures both the command that you submitted (for example, **show ip traffic**) and the device's response (actually, the response from the session running on the device). The command/response pair (and some additional information like timestamp, prompt information, and the session and action identifiers) make up a captured item. As soon as the device responds, iTest adds the captured item as a row in the appropriate session in the Capture view.

The Capture view displays "today's" captured items as they occur with the most recent item at the bottom of the list. You have the option to view the list of captured items grouped by session (as shown in the example — click **Group By Session/Time**).

Tip Double-click a tab to maximize the view. Double-click it again to minimize it.

Each top-level row represents one session, marker, or comment. The cells for a session row are populated if all captured items in the session share the same value for the cell. The cells are blank if the steps have different values.

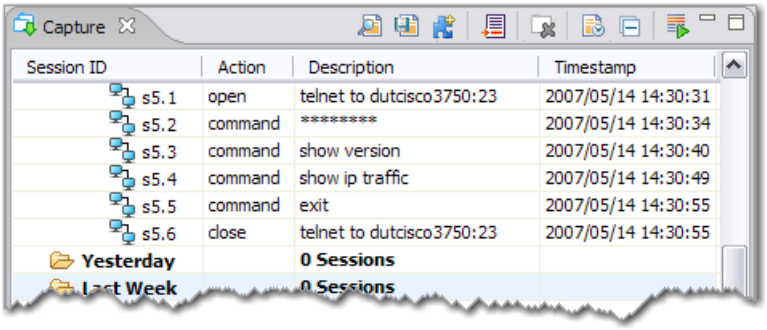
- You can replay captured items or whole sessions by selecting them (use Ctrl-Click and Shift-Click for multi-select) and then clicking **Replay Selected** . You also have the option to replay selected captured items by dropping them into an active session window.
- The Capture view is a log of your session; it is not the place to edit steps. To edit steps, save the captured items into a test case and use the Test Case editor to modify the steps as needed.

- Save selected items as a Capture report by clicking **Save As Capture Report**
- Save selected items as a procedure by clicking **Save As Test Case**
- Insert comments by typing them into the Capture Comments view.
- All responses are captured, but responses do not appear in the Capture view. You can view responses in either the Response view, the Capture Report view, or the Capture report. Double-click an item in the Capture view to display the response in the Response view.

Reviewing a session

Expand a session row to display each captured item in the session. In this example, iTest captured:

- The Telnet session **open** action for session **s5**
- The password that you submitted (masked by ******** characters)
- The **show version** command that you submitted
- The **show ip traffic** command that you submitted
- Your **exit** command and the resulting session **close** action:



Tip You can set preferences for capture. See “Setting preferences for the Capture view” on page 111.

Session ID / Action ID	The value in the **Session ID** column identifies a captured item uniquely by the combination of Session ID and Action ID. The first session is named **s.1**, the second session is **s.2**, and so on. For each session, the actions are numbered **1**, **2**, and so on. So, **s.3.5** represents action **5** in session **s.3**. **Note** The following exception applies: Before assigning the next Session ID, iTest checks to see whether a session that is still running already has the ID (perhaps it started yesterday or earlier today). iTest skips the duplicate ID and tries the next.
Action	The Action identifies the open and close actions as well as actions you perform with sessions, for example: • Commands to the CLI interface on a device (for example, **set routes**) • Web-based actions like button clicks or filling in and submitting a form field • Performing a get or set on an SNMP MIB • Submitting a command to a Tcl session • See “Actions” on page 239.

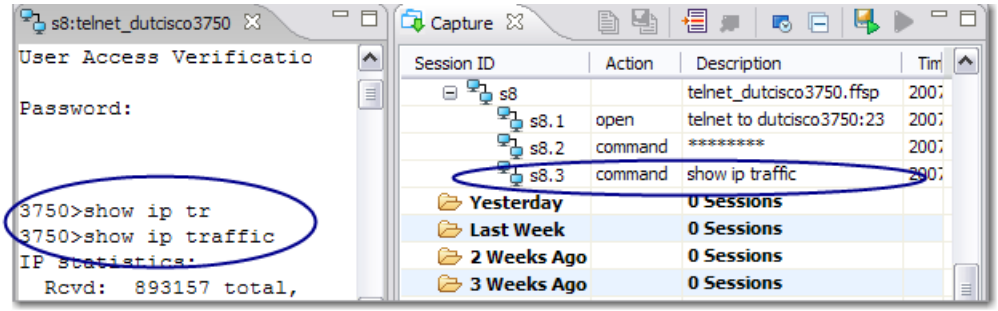
Description	The **Description** column displays the description of the action that you took, for example the text of a command or the identifier of the button that you clicked on an HTML page. For actions of type **open**, the Description is the name of the session profile used to start the session. If the session was started without a session profile, then the Description is the combination of the session type and the device IP address or hostname (that is, the name that iTest creates for the session profile that it saves in the **recent** folder). For actions of type **command**, the **Description** is the text of the command (for example, **set routes**). When a session row is collapsed, its Description cell displays the Description for the session's **open** step.
Timestamp	Date and time timestamp in **yyyy/mm/dd hh:mm** format. For example, 2007/07/04 15:42 For sessions: The date and time that the session opened. For steps: The date and time that the step started.

Using keyboard shortcuts

For CLI sessions, iTest "cleans up" any keyboard shortcuts like backspace, tab-completion, and up-arrow keystrokes so that the command text in the Capture view reads as if you typed it fully and correctly. (You can configure capture cleanup while defining the session profile properties.)

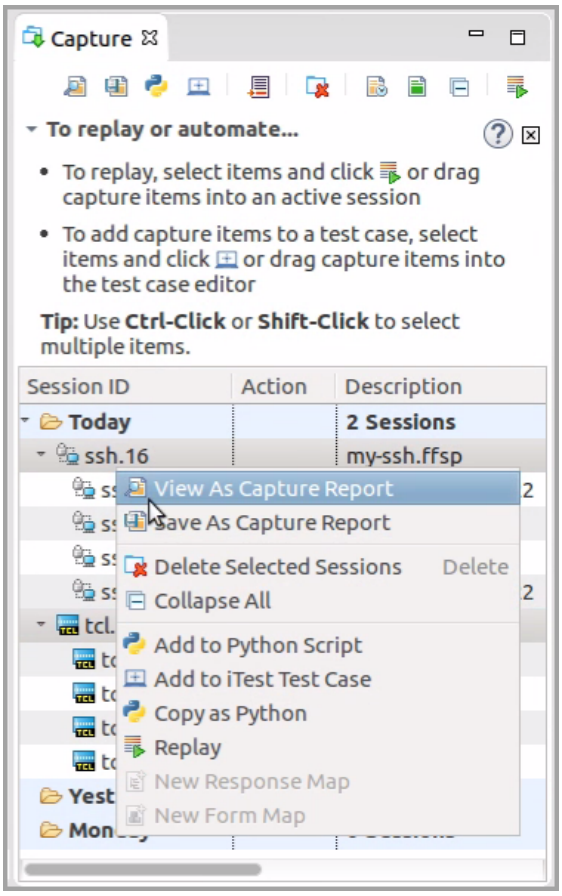
In the following example, iTest captured the **show ip traffic** command correctly, even though we actually typed **show ip tr**<tab>. iTest discarded the intermediate **show ip tr**<tab> form of the command that we actually typed and uses the **"**fully-typed**"** form of the command. This makes it easier to read test cases.

In this example, the command completion form of the command was deleted (**show ip tr<tab>**). Only the "cleaned up" version of the command ("**show ip traffic**") is captured.



When in doubt, right-click

Most of iTest's editors and views include context (right-click) menus for common operations (most toolbar items appear in the context menus). For example, here is the context menu in the Capture view:



What can you do with items in the Capture view?

- View or Save selected captured items or captured sessions as a ***Capture report*** in the Capture Report editor. (You can replay Capture reports.)
- Delete selection sessions
- Toggle to expand or collapse the captured steps.
- Add to Python Script. Includes the selected open steps and the import commands. Where applicable, includes the topology used. See Chapter 59, “Python Script Generation”.
- Add selected sessions as in a iTest Test Case.

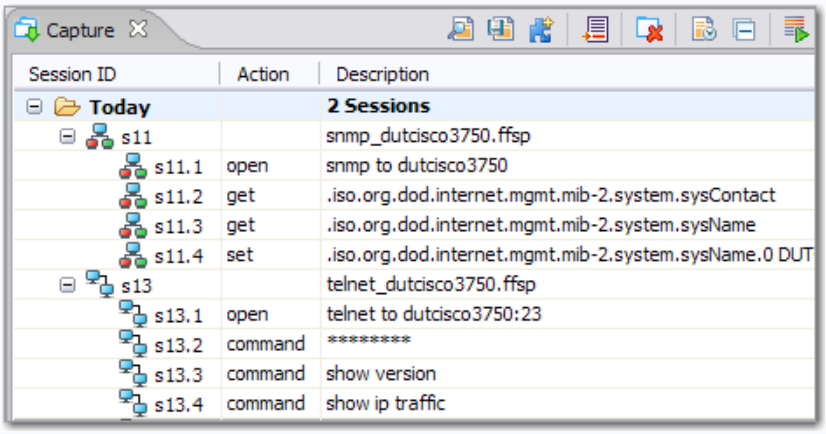
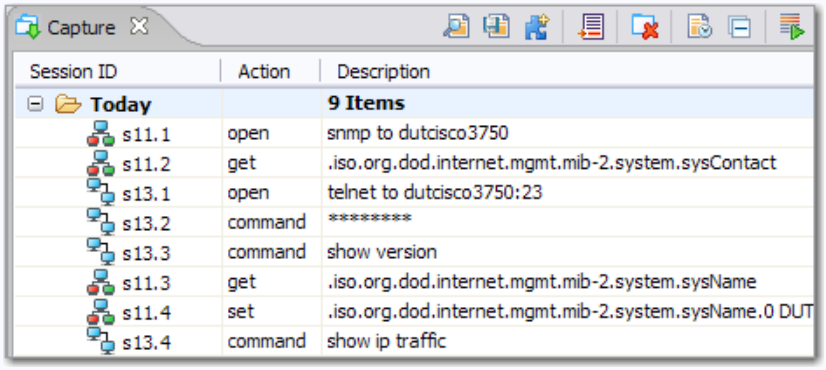
  When you save a captured session as a test case, each QuickCall that you performed during the manual session becomes a ***single step*** in the test case (regardless of how many actions the QuickCall actually performed). This improves readability, portability, and consistency. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.
- Copy as Python. iTest will render the selected steps (open steps, quickcall steps, and native command steps) in Pyhon syntax and copy the line(s) to the clipboard. The contents of the

clipboard includes only the script lines associated with the selected steps. You may insert these as appropriate into your Python script. See Chapter 59, “Python Script Generation”.

- Replay captured items or whole captured sessions by selecting them in the list and dropping them into an active session or by clicking the **Replay Selected** .
- All responses are captured, but responses do not appear in the Capture view. You can view responses in the Capture report. Double-click an item to display it in the Response view.
- The Capture view is a log of your session; it is not the place to edit steps. To edit steps, save the captured items into a test case and use the Test Case editor to modify the steps as needed.
- Use the Capture Comments view to add a comment to the current captured items list (for example, a note to a coworker describing the session's result).
- Add a marker to visually separate portions of the list.

Capture view toolbar

	View the selected items in the Capture Report editor. This enables you to view the captured items in greater detail and gives you the option to save the items as a Capture report. You can view one or more captured items or one or more sessions. Selected comments and markers also appear in the report.
	Save the selected items as a Capture report. You can save one or more captured items or one or more sessions. Selected comments and markers are also saved.
	Add to Python Script. Create a new Python Script with the selected captured items or add captured items to an existing Python Script.
	Save the currently selected items as a iTest Test Case. You can save one or more captured items or one or more sessions. Selected comments and markers are also saved.
	Copy as Python. iTest will render the selected steps (open steps, quickcall steps, and native command steps) in Pyhon syntax and copy the line(s) to the clipboard. You may insert these as appropriate into your Python script.
	Insert a marker into the list of captured items after the most recent item. Markers help you to visually break up the capture log.
	Delete the selected sessions, comments, and markers. While preparing to save sessions as a Capture report or procedure, you may want to remove particular sessions, comments, or markers from the Capture view. Select the items (use Ctrl-click and Shift-click to select multiple items). Click **Delete Selected Items** . You can delete all sessions in one or more folders (**Today**, **Last Week**, and so on) by selecting the folders and clicking . **Note** This action does not delete selected captured items. Only sessions (and their included captured items), comments, and markers are deleted.

	Group by Session or **Group by Time**. Use this button to toggle between the following ways of viewing captured items in the Capture view: • Grouped by session so that all items from a particular session appear under the appropriate session heading. Sessions are ordered by the timestamp of the open Action for the session.  • In the order in which the captured items occur, without regard to the items' sessions (one item from one session and another item from another session mixed in a single chronological list). 
	Collapse all open folders so that only the top-level folders (**Today**, **Last Week**, and so on) appear in the view.
	Replay the selected captured items or any mix of the following items: • Selected captured session or group of captured sessions • All captured sessions in selected folders

- When you expand a session row, the view displays one row for each captured item in the session. Click **expand row** ⊞ to view the captured items that make up the session. The captured items' properties are not editable.
- When you select an item, iTest updates the Response view to reflect the selection.
- Comments are identified by the series **c1**, **c2**, **c3**, and so on. (To add a comment after the latest captured item, type the text into the Capture Comments view and then press Enter.) When you save a comment to a test case, it is saved as a **comment** step.
- Markers are identified by the series **m1**, **m2**, **m3**, and so on. (To add a marker after the latest session, click **Insert Marker** in the toolbar.) When you save a marker to a test case, it is saved as a **comment** step.

Order of sessions in the Capture view

When you specify **Group By Session**, sessions are ordered by the timestamp of the open Action for the session. Captured items within each session appear in timestamp order.

When you specify **Group by Time**, a single list of captured items appears in the order of capture.

Comments and markers appear in chronological order in the list.

Today	Since midnight today.
Yesterday	Between midnight yesterday and midnight today.
Last Week	Between midnight on the Sunday before last and midnight yesterday.
2 Weeks Ago	Between midnight the second Sunday before last and midnight on the Sunday before last.
3 Weeks Ago	Between midnight the third Sunday before last and midnight on the second Sunday before last.
3 Weeks Ago	Between midnight the fourth Sunday before last and midnight on the third Sunday before last.
Older	Before midnight on the fourth Sunday before last.

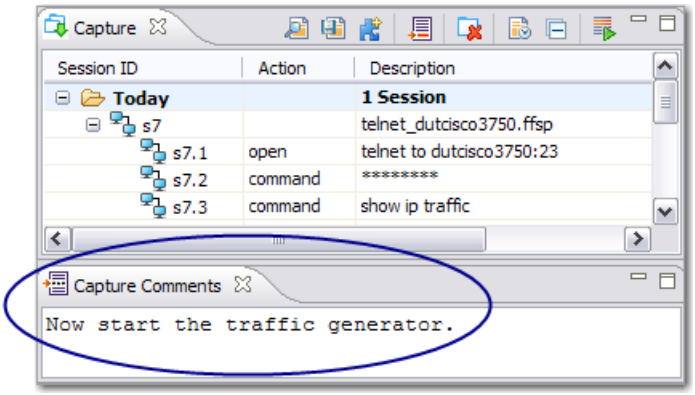
Inserting comments into the Capture view

All captured items appear in an ordered list in the Capture view. At any time, you can add a comment to the end of the current list of captured items (that is, after the most recent item). Comments enable you to note important items or to clarify aspects of the captured steps to other reviewers.

- Comments are added at the same level as the captured items.
- Comments have Action type **comment** and are identified by the series **c1**, **c2**, **c3**, and so on.
- Comments are not replayed.
- Comments are saved when you save the captured items into a Capture report or into steps in a test case.
- When you save a comment from the Capture view as a step in a test case, it is converted into an EXEC comment action.

Adding comments

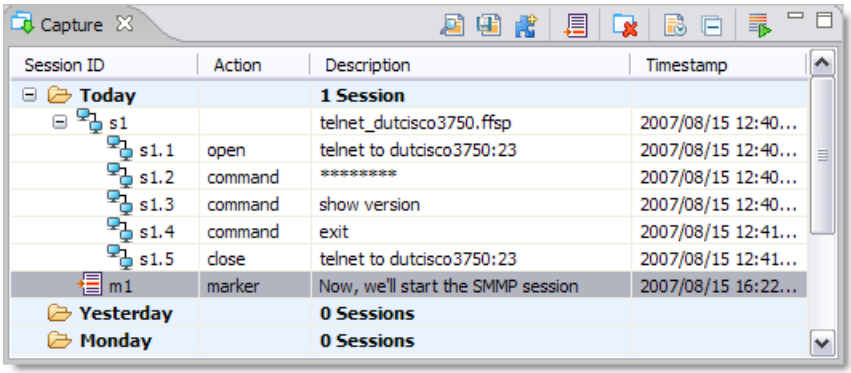
Type the comment into the Capture Comments view and press the **Enter** key. iTest adds a comment after the latest captured item (for Today) in the Capture view.



Markers

All captured items appear in an ordered list in the Capture view. At any time, you can add a marker to the end of the current list. This enables you to visually separate important items.

- Markers are added at the same level as sessions.
- Markers are not executed.
- Markers have Action type **marker** and are identified by the series **m1**, **m2**, **m3**, and so on.
- Markers are saved when you save the captured items into a Capture report. This enables you to annotate Capture reports to set off sections of steps visually.
- When you save a marker into a test case, it becomes a **comment** step.



Inserting markers into the Capture view

All captured items appear in an ordered list in the Capture view. At any time, you can add a marker to the end of the current list.

When you save a captured session into a test case, makers are converted into **comment** actions.

Adding markers

Click **Insert Marker** in the Capture view toolbar. iTest adds a marker after the last captured item in the latest session in the view.

A cursor appears in the **Description** cell for the marker. Type text to associate with the marker.

Copying/pasting markers

To make a copy of a marker in a new location in the list of captured items, select and copy (Ctrl-C) the marker. Select a new location and press Ctrl-V. The copied marker appears after the selected captured item.

Deleting sessions from the Capture view

While preparing to save captured sessions as a Capture report or as steps in a test case, you may want to remove some sessions from the Capture view.

To delete sessions from the Capture view

Select the sessions (use Ctrl-click and Shift-click to select multiple items). Click **Delete Selected Items**. Selected markers and comments are also deleted.

Setting preferences for the Capture view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Views > Capture View.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Views > Capture View

Display replay tips in Capture view	Check the box to display help at the top of the Capture view. Default: unchecked
Manual Step	Click to seelct the color of the manual steps. Defail is

Commands

Note This section provides a quick overview of commands from the perspective of capturing interactive sessions. For a full discussion, see Chapter 17, “iTest Commands”.

CLI sessions (command-line interface) are the terminal-based sessions; any session where you type commands at a prompt in a command line. iTest supports the following CLI terminal-based session types:

Command Prompt

HTTP

Process

SSH

Syslog

Tcl Shell

Telnet

Many Traffic generator device session types

Wireshark

As you perform interactive tests in CLI sessions, iTest captures the command and the response. When you save the steps into a test case, the **Action** type for the saved steps is **command**.

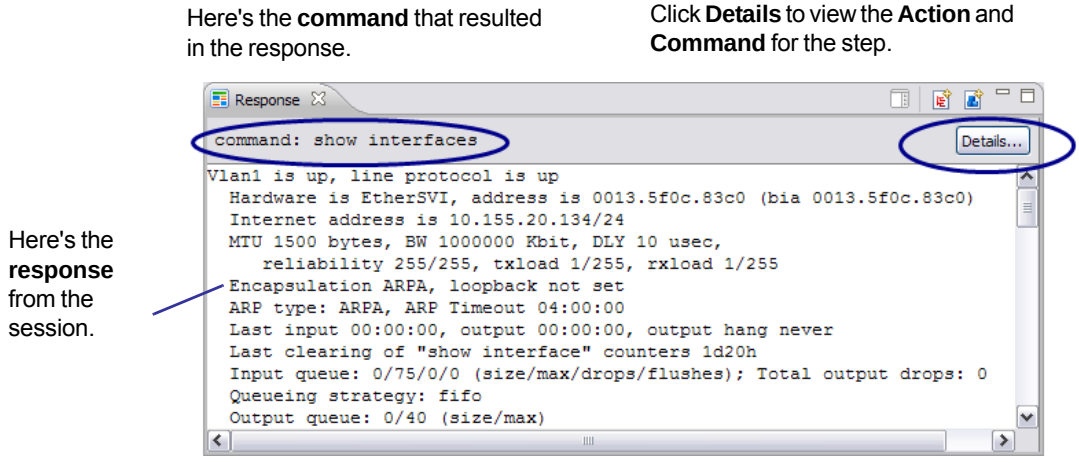
Non-CLI sessions use a different kind of command, as described in Chapter 94, “Web sessions (Obsolete and Deprecated)” and Chapter 82, “Swing Sessions (Obsolete and Deprecated)”.

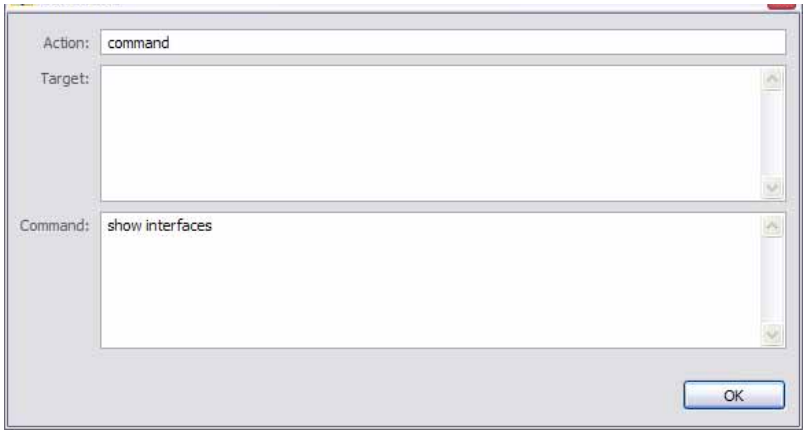
Let's compare how you work with commands while capturing a session and while editing a test case:

Commands in interactive sessions

You send commands to a session and the session responds. (A **command** is a particular type of **Action**.)

Here's an example of a **show interfaces** command that was captured during a Telnet session (as displayed in the Response view):





The **Action** is **"command"**.

The value of the **Command** property is **show interfaces**.

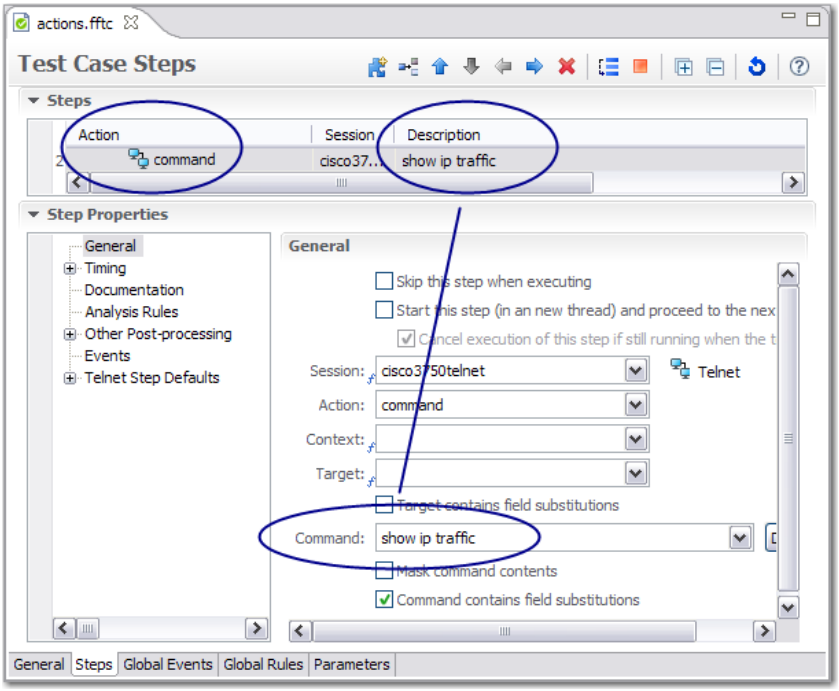
(There is no **Target** because this is not a browser-based step.)

For web-based actions, the **Command** is an optional argument for the **Action**. For example, the argument for the **setText** Action is the text to type into the text box (therefore the value of the **Command** property is the text that is typed).

Most Web **Actions** do not require **Command** values. For example, the **click** Action requires a **Target** (the name of the button to click), but not a **Command**.

Commands in test cases

For CLI sessions, when you are working in the Test Case editor, you define a command by selecting an **Action** of type **command**, and then providing the text of the command in the **Command** property (or the **Description** cell) for the step.



Changes that you make to the **Command** property are reflected in the **Description** cell.

and ...

Changes that you make in the **Description** cell are reflected in the **Command** property (and in other properties in the case of other session types).

There are only two **Actions** that are specific to CLI steps in a test case (the other actions in the list appear for all session types):

command: A **command** action submits a command to the session.

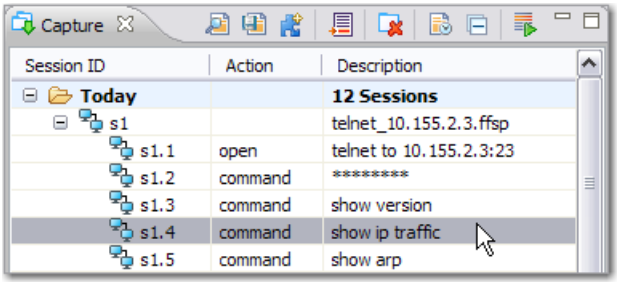
break: A **break** action is special. It sends a break character (typically Ctrl-C, configurable in the session profile)

There is one **Action** that is specific to traffic generator session types: **configure**

Captured items

In both manual (interactive) sessions and automatically executed test cases, a session typically returns a response to each command that you send. iTest captures the command, response, and other identifying information (like the session and action identifiers and timestamp) as a ***captured item***.

The Capture view displays basic information about each captured item and lists each item in the order in which it was captured. In this example, we have selected a single captured item (the **show ip traffic** command that we submitted to the device).



You can save any or all of the captured items that currently appear in the Capture view as a Capture report or a procedure in a test case. Capture reports preserve and display all of the information about all captured items.

The Capture view is a log of your session; it's not the place to edit steps. To edit steps, save the captured items into a test case and use the Test Case editor to modify the steps as needed. Captured items include the session's response, but the Capture view does not display responses. To view a response, open the Response view or view the items in a Capture report. Double-click an item to display it in the Response view.

On startup, iTest will automatically discard old captured sessions when the size of the capture database exceeds a certain limit. You can change this limit using **Window > Preferences**.

Open and Close steps

Note This section provides a quick overview of the **open** and **close** commands from the perspective of capturing sessions. For a full discussion, see Chapter 17, “iTest Commands”.

As you work in iTest, you'll notice that the open and close actions that you perform during interactive sessions (starting a session, exiting the session) become **open** and **close** actions in test cases (when you save the sessions as procedures in test cases).

Let's compare **open** and **close** actions while capturing a session and while editing a test case:

Open and Close in manual (interactive) sessions

When you start an interactive session (for example, by double-clicking a session profile), iTest captures an **open** action. When you close a session (for example, by submitting an exit or quit command), iTest captures a **close** action.

Open items in Capture reports or captured sessions

To ensure that you can replay a Capture report or captured session by double-clicking it or by clicking , you must include the **open** action. If a session or report does not include the **open** action, then, to execute it, you must drop it into an existing session of the appropriate type.

Open and close steps in test cases

Each session in a test case starts with an **open** action as its first step. See “The open action: Start a session” on page 241.

The **Command** property for an **open** step must be a full Spirent URL (typically, the URL of a session profile, but other URL types are supported, as described in “The open action: Start a session” on page 241. (iTest populates the URL when you save a captured session as a procedure.)

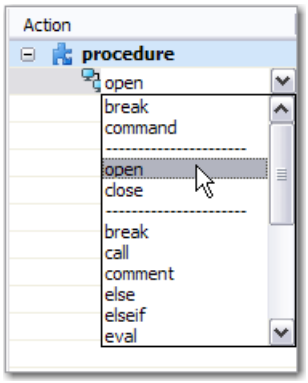
There are special properties for an **open** step.

Each step in a test case that interacts with a session must specify the name of the session in the **Session** property.

The last step for a session is typically a **close** step.

When you add the first step to a new procedure, iTest inserts an **open** step. You complete the step by selecting a session profile from the drop-down list in the **Description** cell.

Notice that **open** and **close** actions appear in a separate group in the **Action** drop-down list. This indicates that **open** and **close** are special: All sessions must start with an **open** step and typically include a **close** step. See “The open action: Start a session” on page 241 and “The close action: Close a session” on page 243.



Capture Comments view

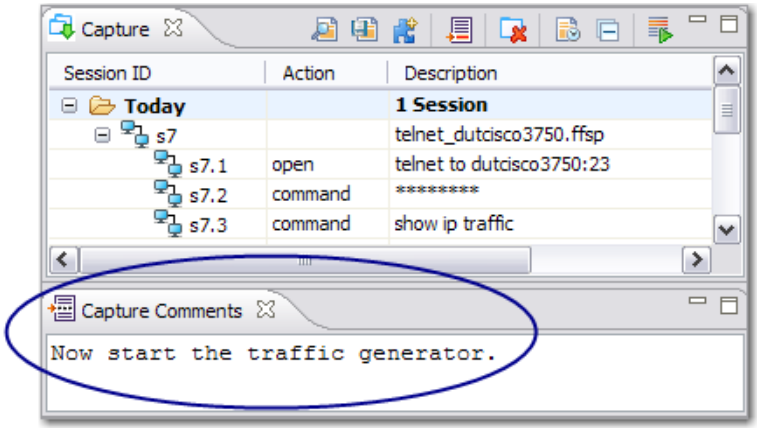
All captured items appear in an ordered list in the Capture view. At any time, you can add a comment to the end of the current list of captured items.

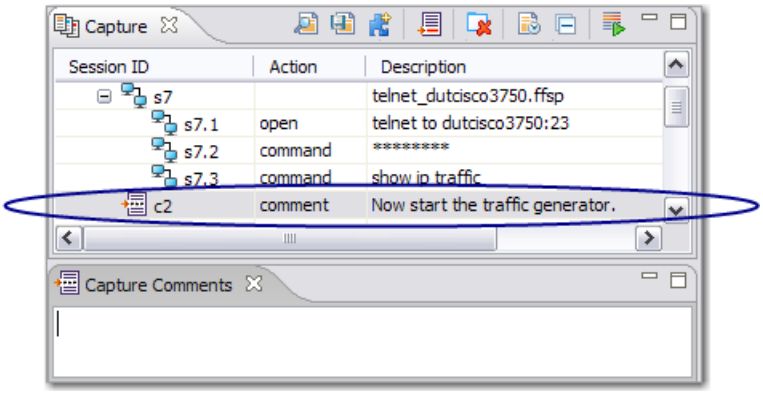
Comments are saved when you save the captured items into a **Capture report**. Use comments to identify important items in Capture reports or to clarify aspects of the test to other reviewers. When you save captured items into a test case, comments are saved as **comment** steps.

Note Double-click a tab to maximize the view. Double-click it again to minimize it.

Adding comments to the Capture view

Type the comment into the Capture Comments view and press the Enter key.





About capturing and defining prompts

We humans know that the text "**C:\>**" represents a prompt, but how does iTest know that both **C:\Temp>** and **C:>** are acceptable prompts? iTest works with prompts in the same way as we humans do: it looks for particular text strings followed by a few milliseconds of idle time on the session channel.

Spirent has predefined some of the most common prompts, however, you might have to define the list of custom prompts for a particular session profile or device. See Chapter 22, “Prompts (in CLI sessions)” for instructions.

Saving captured sessions or captured items as a Capture report

At any time, you can save selected captured items into a Capture report.

1. In the Capture view, select one or more captured items (use Ctrl-click and Shift-click to select multiple items).
2. Click **Save As Capture Report** (alternatively, right-click the selection and select **Save As Capture Report**).
3. On the **Save Capture Report** dialog box, in the **Container** field, specify the path to the report that you are about to create.
4. iTest suggests a filename. Specify the name of the file. Click **Finish**.

   iTest creates the document and then selects it in the iTest Explorer.

View Before Saving

You also have the option to view the current set of captured items in the Capture Report editor before saving

1. In the Capture view, select one or more captured items (use Ctrl-click and Shift-click to select multiple items).
2. Click **View As Capture Report** . The Capture Report editor opens and displays all selected items.
3. To save the displayed items as a Capture report, click **Save As** in the main toolbar.

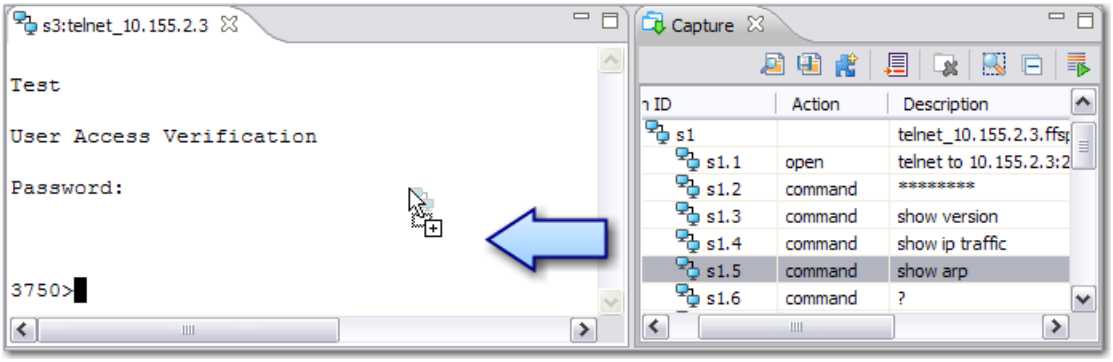
Saving captured sessions as a Capture report

1. In the Capture view, select one or more sessions (use Ctrl-click and Shift-click to select multiple items).
2. Click **Save As Capture Report** (alternatively, right-click the selection and select **Save As Capture Report**).
3. On the **Save Capture Report** dialog box, in the **Container** field, specify the path to the report that you are about to create.
4. iTest suggests a filename. Specify the name of the file. Click **Finish**.
5. iTest creates the document and then selects it in the iTest Explorer.

Replaying captured items

If the group of captured items is all in the same session and *does not* include an **open** item, then drop selected captured items from the Capture view into an active session window of the

appropriate session type. The items execute immediately. (You cannot include session **open** items when dropping captured items.)



Tip For a single item, Shift-drop the item into an active session window of the appropriate session type. iTest pastes the command into the session window, but does not execute it. This option enables you to edit the command before submitting it.

Alternatively, if a session window of the appropriate type is active, then select the items and click the **Replay Selected Items** on the Capture view.

If the group of captured items is all in the same session and *does* include an **open** step, then select the items and click on the Execution view.

To execute a group of captured items from different sessions, the **open** step for each session must precede the other steps in the session.

Tips

- Because iTest always captures commands and responses, it captures the commands and responses that are being executed. This means that you can create a new (more complex) Capture report or session that results from executing multiple original Capture reports or captured sessions.
- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and Paste them into an active session. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. When you save a QuickCall to a test case, iTest saves it as a single step, regardless of how many steps the QuickCall executed (this makes the test case more modular and easier to read). See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Capture reports

Capture reports and the Capture Report editor

Capture reports describe exactly what happened during capture (either interactive sessions or executed test cases) — actions that you took, the response from the session or device, session open and close actions, and step timing and other session information. You create Capture reports by saving selected captured items or sessions as Capture report documents. Use the **Details** page of a Capture report (as described in “Capture Report editor, Details page” on page 120) to view or execute a report.

Capture information is stored automatically by iTest in a database embedded in the iTest workspace. Therefore, some capture reports that you view in an editor will not have a file associated with them on your disk. You can, however, save capture reports into files and open those files using the same iTest capture report editor.

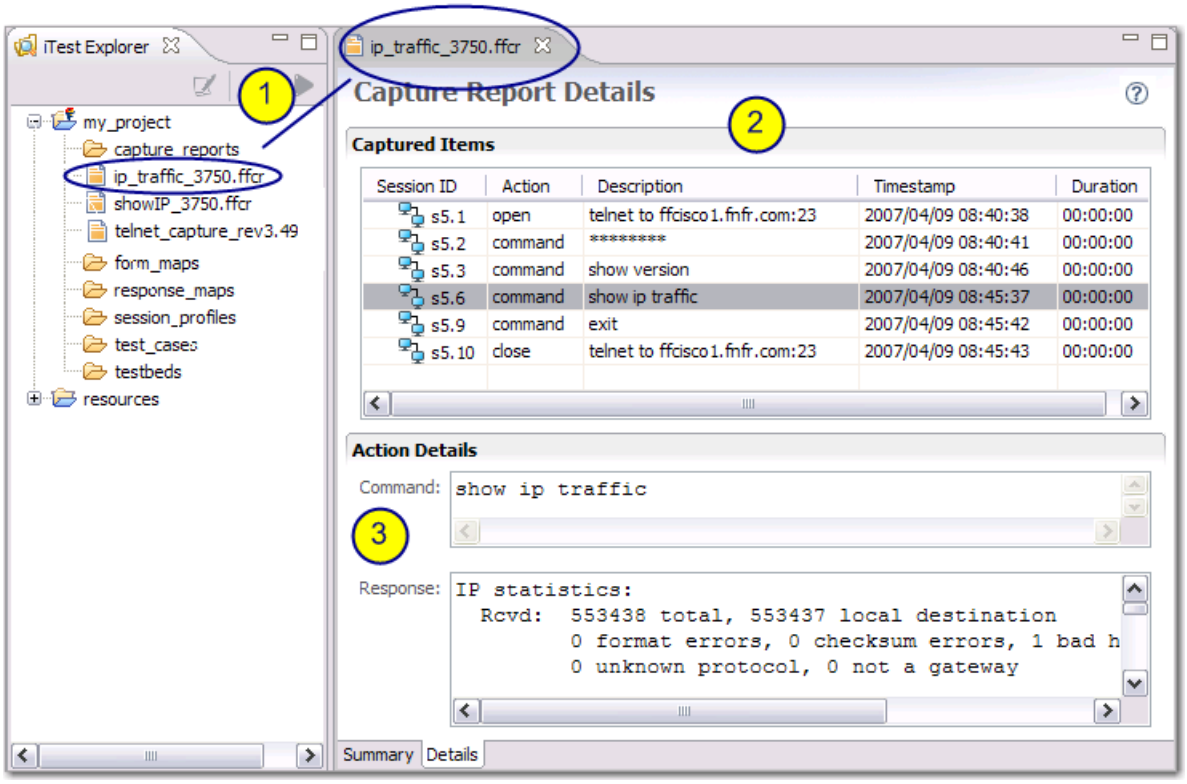
On the **Summary** page of a Capture report (as described in “Capture Report editor, Summary page” on page 122), you can view capture statistics and add notes for coworkers. The **Details** page of the Capture Report editor (shown in the example) enables you to view the details for each captured item.

You may want to share Capture report documents with others; for bug reports, test case documentation, and so on. You can use Capture reports to:

- Generate detailed bug reports
- Copy a set of steps in the report and then paste them (for example) into a Test Plan document
- Prepare reports proving that operation worked as expected (or not)
- Execute the captured steps, that is, have iTest execute the captured steps automatically just as you had originally executed them

Capture Report editor, Details page

The Capture Report Editor opens to the **Details** page.



1. Capture reports are typically saved in the **capture_reports** folder in iTest Explorer. Double-click a report to open it in the editor.
2. Capture reports are special. You can view the capture results as you would expect, *and*, in addition, you can replay a report just like a script. While viewing a Capture report in the editor, click **Execute in new window** .
3. Select a captured item to view the **Command** and **Response** for the item. You cannot edit the report, but you can copy this text to paste into other documents.

Captured Items

Each item in the Captured Items list (default sorting by timestamp) represents a single captured item and the information associated with its execution.

Session ID	For devices with more than one session attached or for multiple captures that use the same session profile: To create the Session IDs that appear in the **Session** cells in the Test Case editor and in test reports, Spirent iTest uses the combination of **Session name** from the session profile (for example, **myDUT**) and a unique session number for the day. (for example, **myDUT.1** and **myDUT.2**). If the session profile does not specify a **Session name**, then Spirent iTest uses the filename of the session profile in its place. The unique identifier for a captured item is the combination of its Session ID and the action number. The first action in each session to start is assigned **1**, the second action is **2**, and so on. **Example** For the third and fourth captured items on session **myDUT.1**, the identifiers are **myDUT.1.3** and **myDUT.1.4**
Action	You perform actions on a session — send a CLI command action to a Telnet session, click a link to perform a **get** action in an HTTP session, send **get** or **set** actions regarding a MIB variable in an SNMP session, send a command to a Tcl session, and so on. For example, the type of Action that you take in a Telnet session with a device is called **command.** When a session starts, iTest sends the first action for the session, an open action. When a session ends (for example, when you send an **exit** command in a Telnet session), iTest sends the last action for the session, a **close** action. The actions that you perform occur between the **open** and **close** actions.
Timestamp	The date and time that the items were captured in **yyyy/mm/dd hh:mm** format.
Duration	The time that it took the action to run to completion during capture. For CLI commands, this is the return of the prompt. For actions on a form, this is the return of the full response (HTML page or form). **Note** This may include time that the original person was away from the keyboard, so it does not necessarily represent the fastest execution time.

Action Details

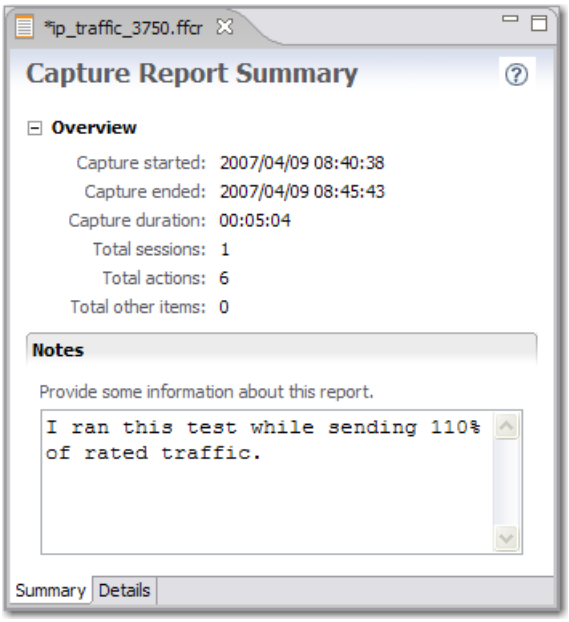
Command	The command sent to the session. You can copy text from this section.
Response	The session's response. You can copy text from this section.

Capture Report editor, Summary page

The **Overview** provides statistics on the capture.

Type notes into the text box to communicate with coworkers. The notes are saved with the report.

We're viewing the **Summary** page of the report.



Overview section

The Overview section of the Summary page displays basic information about the captured items (start and finish timestamps, total sessions, actions, items, and so on). You cannot change the information on the page. As with all aspects of the report, the information is stored in the document in XML format.

Notes section

You can type notes regarding the overall report (for example to describe what happened during capture or to add a note to a coworker). The text is saved when you save the report and appears here when anyone opens the report.

Note In addition to the text in the **Additional Information** text box, you can, during capture, insert comments into the list of captured items. See “Inserting comments into the Capture view” on page 109.

Viewing a Capture report

You can open a Capture report in any of the following ways:

- In the iTest Explorer, double-click a report.
- Right click a report in the iTest Explorer and then select **Open**.

- Select a report in the iTest Explorer and drop it into the Editor area (the area in which the **Start a New Session** page normally appears).

The Capture Report Editor opens to the **Details** page.

Viewing captured items in the Capture Report editor

At any time, you can view the details of selected captured items in the Capture Report editor.

1. In the Capture view, select one or more captured items (use Ctrl-click and Shift-click to select multiple items).
2. Click **View As Capture Report** in the Capture view toolbar. The Capture Report editor opens and displays all selected items.
3. To save the Capture report, click **Save As** in the main toolbar.

Setting preferences for capture

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > General > Capture**.

Note The disk space and cache settings will take effect after you restart.

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Spirent > General > Capture

Maximum disk space to allocate for storing captured sessions	The Capture process holds a large number of captured items, up to a specified limit. When the data exceeds the limit, iTest deletes a certain number of the oldest captured items to make room for new captured items. The default setting allows for most users' needs. Default: 1000
Maximum number of captured steps cached in memory	The Capture process holds information on captured steps in memory, up to a specified limit. When the step count exceeds the limit, iTest deletes the oldest captured step to make room for a new captured step. The default setting allows for most users' needs. Default: 1000

Suppress capture of sessions started via execution	Select one of these three options. **Note** Your selections will take effect when you click **Apply** (there is no need to restart iTest). **Suppress:** The selection ensures that Capture function is not performed during execution, including the manual steps when paused.
	Suppress, if you perform steps interactively while execution is paused then capture: Capture view is populated with all steps you perform (manually) in interactive sessions. That is, no steps will be captured for session during execution, until at least one manual step is performed in the session: • Captures nothing when no manual steps are performed • Captures all/any manual steps performed in the session The session is also captured when you perform steps interactively while execution is paused.
	Do not suppress: Capture all steps during execution, including the manual steps when paused. Captures all steps — whether performed by you in an interactive session or auto-executed by iTest — are captured and displayed in a session in the Capture view.

CHAPTER 7

Test Cases

This chapter includes instructions for several tasks that you will perform in creating, editing, and maintaining test cases. You do all of that work in the Test Case editor. For information on using the tools and options available in the Test Case editor, see Chapter 8, “Test Case Editor”. In addition, see Chapter 20, “Debugging Test Cases”.

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Overview: Creating a test case

Creating a test case by capturing interactive sessions

You can create a test case by typing steps into the Test Case editor, but the easiest way to create a test case is either of the following methods:

- **Direct-to-test capture**. Click Start Capture, perform the steps interactively (manually), and then click Finish Capture. iTest converts the captured steps into an automated test case and displays it in the Test Case editor. For details, see “Saving interactive steps as steps in a test case: The ‘Add to iTest Test Case’ wizard” on page 98.
- **Save a captured session as a procedure.** This method allows you to select any set of captured steps in the Capture view (regardless of when they occurred) and add them to a test case. See “Adding captured steps into a test case or Python Script” on page 100.

In either case, the **Add to Test Case** wizard starts and gives you the option to either:

- Create a new test case and add the steps to it
- Add the steps to an existing test case

Other options for creating a test case

Here are some other options for creating test cases:

- From the Capture view, drop captured items and captured sessions into a test case document
- Copy steps and/or procedures form one test case to another
- Create a new test case document and manually type steps and procedures.

Use any of the following methods to create a new test case document:

- In the iTest Explorer, right-click the intended parent folder and select **New > Test Case**.

- In the main menu, click **File > New > Test Case.**
- In the main toolbar, click **New Document** and select **Test Case**.
- You can copy/paste an entire test case document (for example, to create a test case that is very similar to the original). If you copy a test case, remember that the most recent response to each step (in the most recent execution of the test case) is also copied. This can result in old information in the Response view as you start to work on the “copy”.

Once you have the basics in place, execute the test case and modify as needed. You can skip steps and procedures, use breakpoints or pause execution to check what's happening. You can add flow control like **If** constructs or **For** loops. You can add comments to document steps or procedures.

Editing a test case after you create it: The Test Case editor

This chapter includes instructions for several tasks that you will perform in creating, editing, troubleshooting, and maintaining test cases. You do all of that work in the Test Case editor. For information on using the tools and options available in the Test Case editor, see Chapter 8, “Test Case Editor”.

Tips for creating a test case

To make a test case easier to understand and to debug, add a **comment** step and indent (nest) any number of steps under the comment. You can then skip all of the steps by skipping just the **comment** step. In addition, you can collapse the **comment** step to hide the steps temporarily. See “Tips on working with test case steps” on page 152.

If you find that you use a particular set of steps, procedures, and/or property settings whenever you create a new test case, you might consider creating a test case to use as a template. Then, whenever you create a new test case, the steps and properties of the template automatically appear in the new test case. See “Creating a template for new test cases” on page 142.

About topologies or testbeds when capturing steps into a test case

If you capture several sessions that use different topologies or testbeds, then the resulting test case uses the testbed used by the first session to start as its Local testbed (a property that you set on the **General** page of the Test Case editor).

Storing response text into a file

You might want to save the response for a step to a file so that you can later review the response while debugging your test case or test suite. Here is an example text file for an **ls** command:

```
Here is the directory listing
action:command
session:t1
command:ls
timestamp:2011/02/24 09:56:12
1              Mail          Aquilla_16239   fibonacci.ffts
```

Note For SNMP sessions, you have the option to conserve memory by specifying not to capture the SNMP response and/or SNMP statistics for the response (in the **Step Defaults** properties group). When you configure a step to save a response to a file, the settings are ignored and all SNMP responses and statistics are always written to the file.

◆ To store response text into a file

1 Select the step. In the **Step Properties** section, open the **Other Post-processing > Store Response** property group.

Note The property settings that you can configure here for the step are all inherited from the **open** step for the session. To log the responses for all steps in a session, therefore, configure the properties for the **open** step.

2 Specify the path and name of the **File to write response to**.

3 In the **Response header** box, specify the text that should appear at the beginning of the text of the response (this is the beginning of the file if you uncheck the **Append response to file** property). In the example, the header text is `Here is the directory listing`.

Note By default, substitution of field replacements is disabled for both the **File to write response to** and the **Response header** properties. You have the option to enable substitution so that, for example, the text **[param headerText]** is replaced with the value of the **headerText** parameter. (Field replacements are described in Chapter 28, “Field Replacements”.)

4 Configure additional settings as appropriate:

Append response to file	Check the box to add the response text to the end of the existing file. Uncheck to overwrite the file. **Caution** If you uncheck the box, then the response for a single step can overwrite a file that contains many appended responses. Default: checked
Note All of the following text items will appear before the response text for the current step. The items appear in the listed order.	
Prepend step action	Check the box to add a line of the form **action:**<actionName> Default: unchecked
Prepend Session ID	Check the box to add a line of the form **session:** <sessionID> Default: unchecked
Prepend command body	The command body is the text that appears in the **Command** property (and the **Description** cell) for a step. Check the box to add a line of the form **command:**<commandBodyText> Default: checked
Prepend timestamp	Check the box to add an execution timestamp for the current step. Default: checked

Inserting captured steps into an existing test case

Here's a fast way to add steps that iTest captured during an interactive session:

1. In the test case, set a breakpoint where you want to start adding steps.
2. Execute the test case.
3. When execution stops at the breakpoint, select the current session window (if it is CLI, it will be stopped at a prompt).
4. Enter any commands that you want to add at this point.
5. In the Execution view, click  to complete execution.
6. In the Capture view, select the last session. Click  to save it as a new procedure.

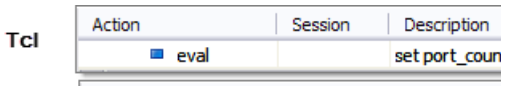
Note When you convert manual steps into test case steps, iTest uses the combination of **Session name** from the session profile and the unique session number for the day (for example, **myDUT5**) to create the **Session ID** that appears in the **Session** cell in the Test Case editor.

iTest interpreter commands in steps

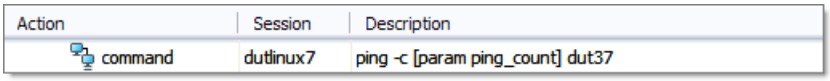
You can use iTest interpreter commands to perform a variety of tasks; some commands set variable values, perform mathematical operations, return information about a device, or parse a portion of a response, others are replaced at runtime by the value of a variable, a parameter value, query results, the full response to an earlier step, a special character, or any of a variety of other values.

Examples

In this example **eval** step, a **set** command sets the value of the **port_count** variable.



The **param** command returns the value of a parameter. In this example, the **param** command in the **command** step is placed inside a field replacement. At runtime, before the step is interpreted, iTest substitutes the returned value for the field replacement (in this example, iTest substitutes the value of the parameter named **ping_count**). So, if the parameter had the value **9**, then the step executes as **ping -c 9 dut37**. (Field replacements are described in Chapter 28, “Field Replacements”.)



For full descriptions of iTest commands, see Chapter 17, “iTest Commands”.

Defining and calling procedures

You can use a **call** step or **CallProcedure** action to execute a local or foreign procedure. See Chapter 10, “Procedures”.

Running child test cases

Create a test case and add other test cases that may be run from within this test case. The test case that contains other test cases is referred to as a Master test case and the included test cases are referred to as slave or child test cases.

Executing a child test case: The ‘run’ action

A **run** step executes the specified test case (the ***child*** test case — sometimes called a foreign or an external test case, from within a ***master*** test case) and optionally passes parameter values. The response for a **run** step is configurable.

- You cannot use the **run** action to execute a procedure — instead, use a **call** step or **CallProcedure** action as described in Chapter 10, “Procedures”.
- You can run child test cases without loading a testbed or topologies for the parent main procedure.
- The child test case executes exactly as if it had been executed using itestcli, except that execution occurs within the current process context.
- The **run** step response can include a table of all child test case executions (either the test case that was executed directly by the **run** or the test cases that were executed indirectly when **run** was executed inside one of the child test cases, and so on).
- You have the option to open a session in the parent test case and then, without closing the session, use the **Shared sessions** property in a **run** step to allow the child test case to use the currently open session. For sessions that take time to start, this feature can save significant time.

Note If you use the **Shared sessions** property, then you cannot enable threaded execution for the step. Conversely, if you enable threaded execution for the step, then you cannot share sessions.

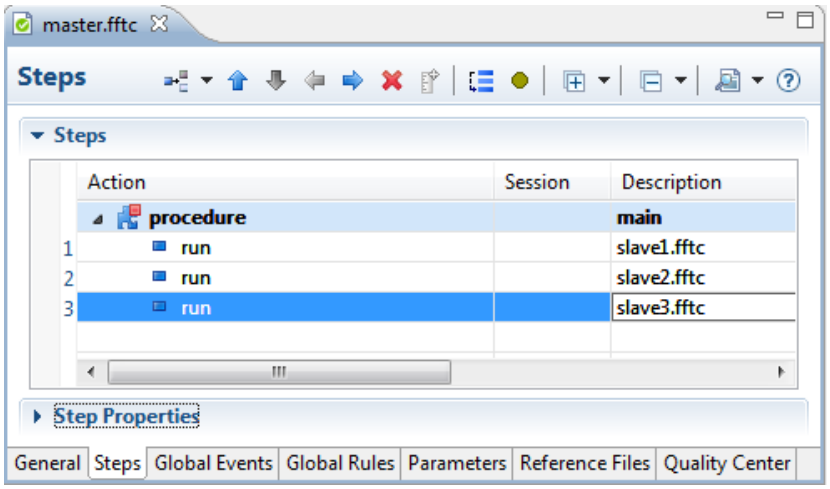
Shortcut: To add a ‘run’ step

While editing a test case, right-click the child test case in the Favorites view and select **Insert step to run this test case**. The **run** step is added after the selected step in the parent/master test case. See “Using the Favorites view to add steps to a test case” on page 299.

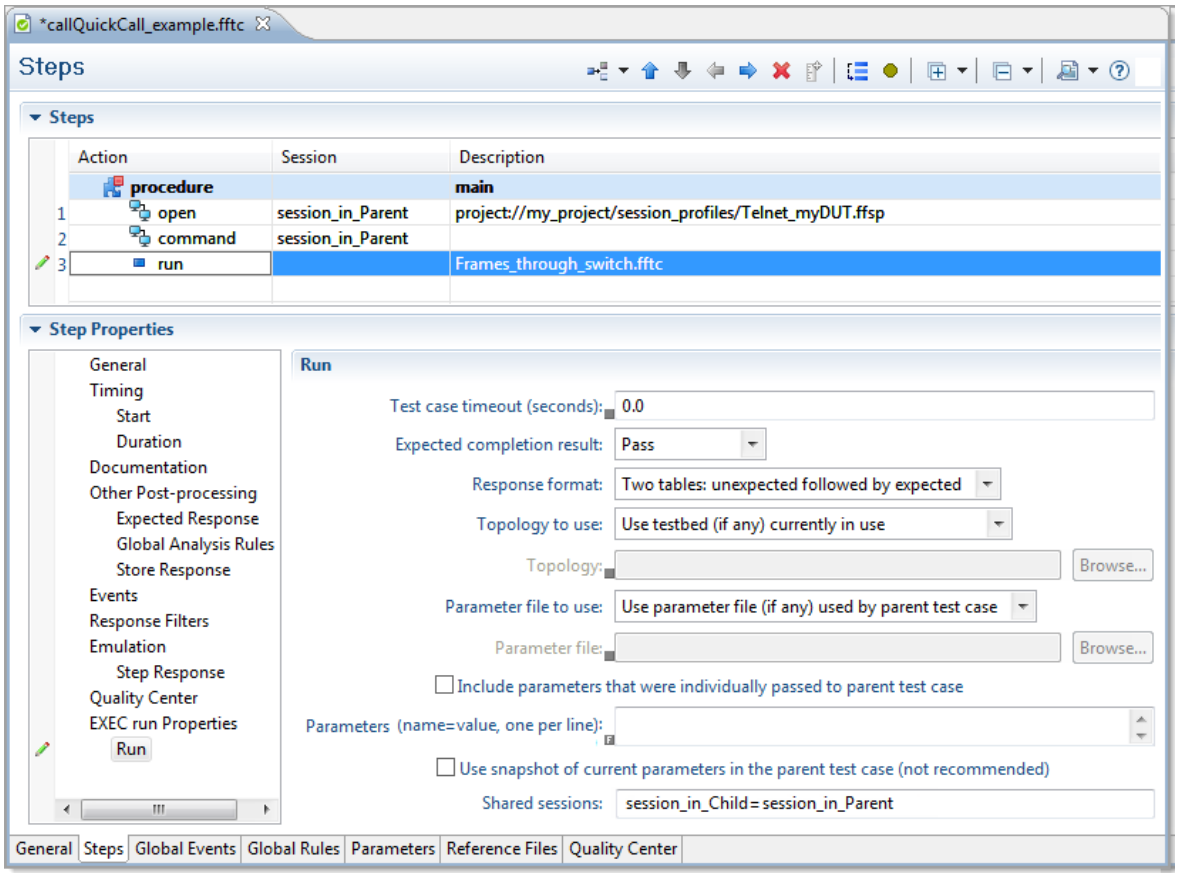
To add a ‘run’ step

1. Create a step with an **Action** of **run**.
2. In the **Description** cell, specify the URI of the test case to run: Click  to open a dialog box that allows you to select a test case either:

- From the list of test cases in the current workspace
- From another location on the file system (typically a test case in an itar file)



3 In the **Step Properties** section, open the **EXEC run Properties > Run** properties group.

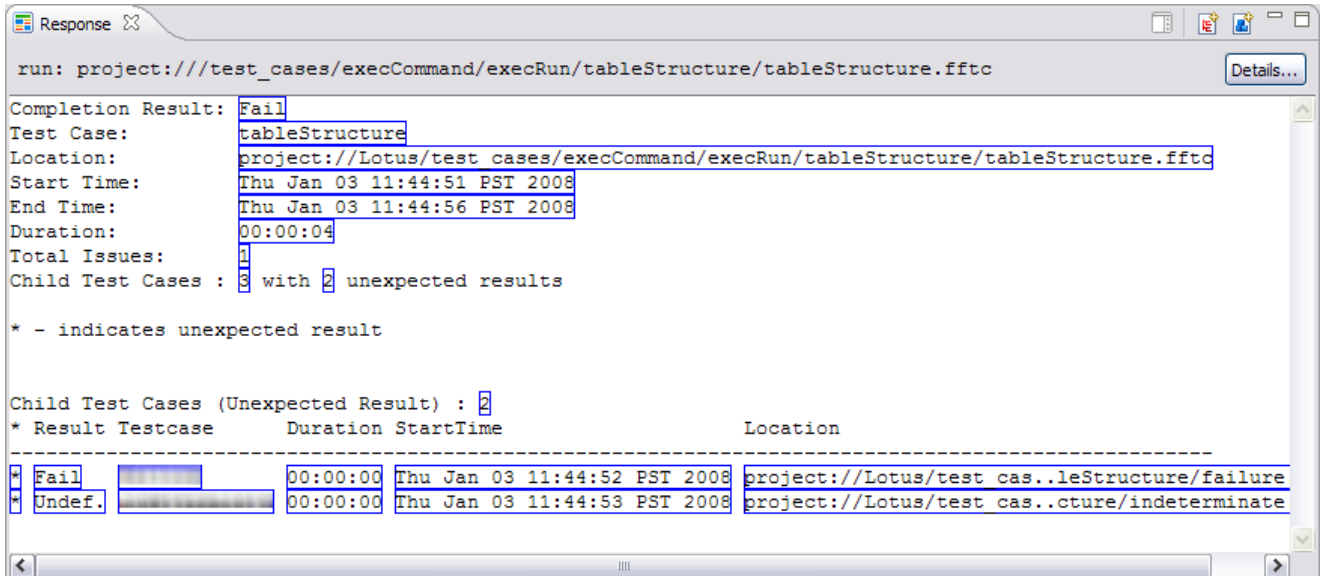


4 Set property values as follows:

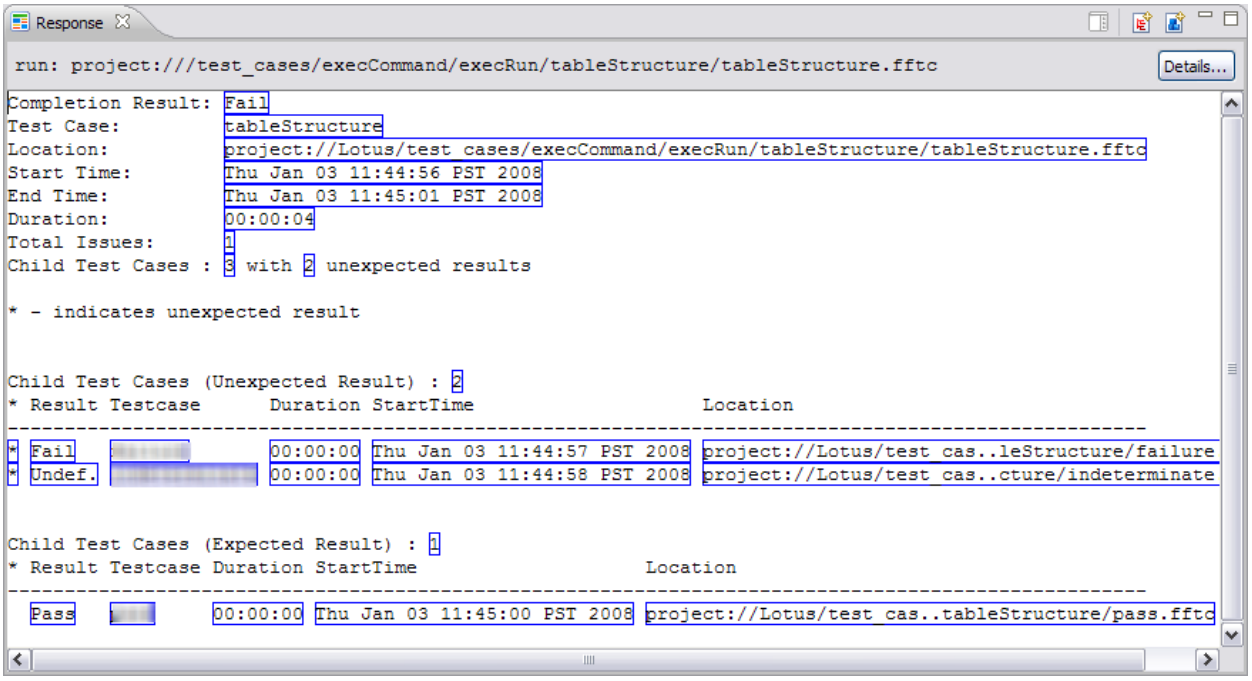
Test case timeout	Specify the maximum time in seconds that the test case is allowed to execute. If test duration exceeds the specified limit, then the test case is aborted and marked **Fail**. The default value of 0.0 indicates no time limit — the test case will “never” time out.
Expected completion result	Specify the result that you expect for the child test case. This property enables you to ensure that the overall test result reflects your preference. For example, if the child test case is a negative test, then, you would expect it to return a **Fail** result to indicate that the test successfully performed the test. In this case, you should set **Expected completion result** to **Fail**. If the child returns a **Fail** result, then, for the parent test case, the **run** step passes.

5 **Response format**: Specify the format of the response to the **run** step. “**Unexpected results**” means any value other than the value that you specified for the **Expected completion result** property. Example response formats:

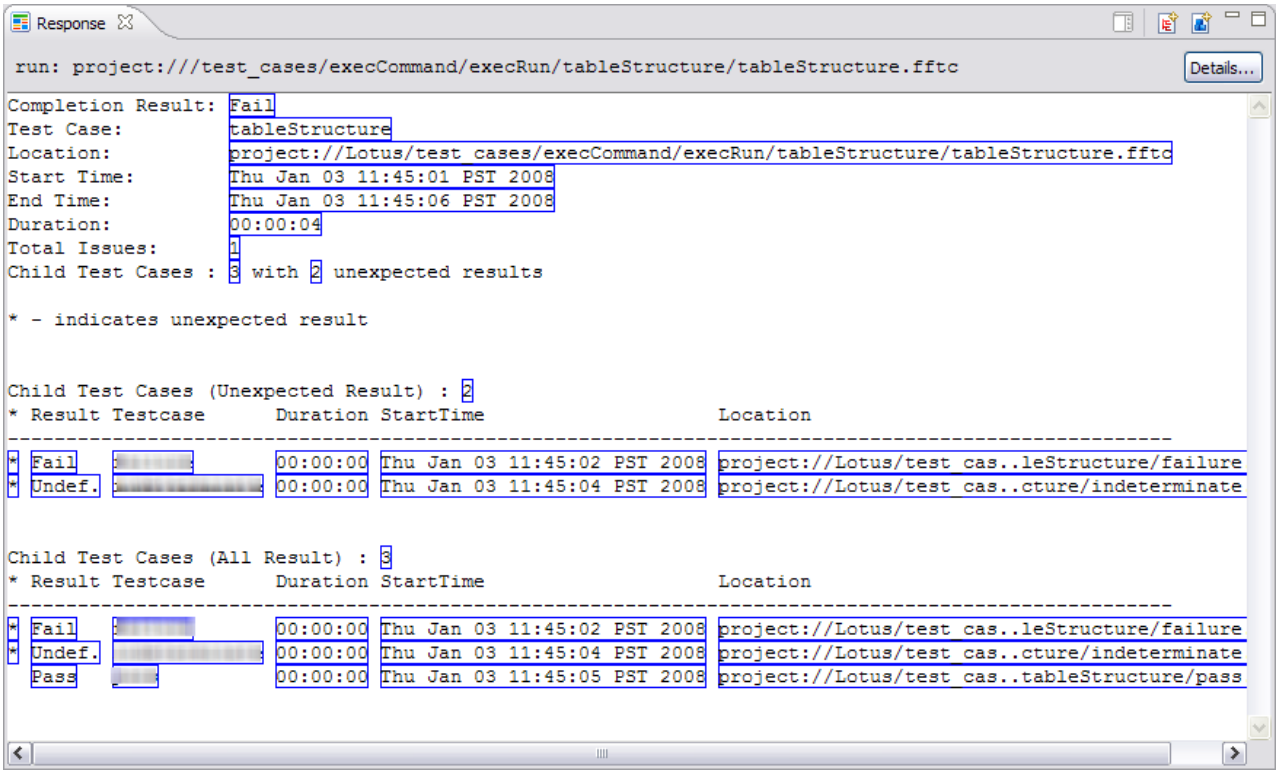
List only tests with unexpected results



Display two tables: unexpected followed by expected



Display two tables: unexpected followed by all tests



6 **Testbed to use:** Optional. Specify the testbed on which the child test case should run. If you do not specify a testbed, then the child test case uses the testbed currently in use by the parent test case.

Use default testbed associated with test case being run	Use the testbed specified on the **General** page of the Test Case editor (for the child test case). This is either: The **Local testbed**, if specified or If no **Local testbed** is specified, then the Global testbed
Use testbed (if any) currently in use	Default. The child test case uses the testbed currently in use for the parent test case.
Use specified testbed	Use the specified testbed to execute the child test case. If you select this option, then you must specify the testbed in the **Testbed** field.

7 **Parameters:** Specify the sources for parameter settings that the child test case should use during execution. In the default configuration, only parameters from the testbed associated with the test case and no others are used.

Parameter file to use	This property determines the source of the parameter file. • **Use specified parameter file**: If you want to use a parameter file, then you must specify the file in the **Parameter file** property. • **Use parameter file from parent**. Use the parameter file specified for the parent test case Default: Use specified parameter file
Parameter file	Optional This property is used when you select **Use specified parameter file** for the **Parameter file to use** property. Specify the parameter file to use while executing the child test case. If blank, then no parameter file is used. Default: [blank]
Include parameters that were individually passed to parent test case	Check the box to cause the child test case to use parameter values as specified in the parent test case. This includes **Advanced Merging** property settings. Default: unchecked
Parameters	Use the specified parameter values while executing the child test case. Parameters that you specify here take precedence over the parameters from any other source. Type only one name=value pair per line. Specify each parameter setting as <name>=<value>, for example, **port=3**.
Initialize using a snapshot of current parameters	**Note** This option is not recommended because it can cause duplicate parameters to be created. In addition, the snapshot may include parameters from other child test cases. Check the box to cause the child test case to use parameter values as currently in use. This includes **Advanced Merging** property settings. Default: unchecked

8 **Shared sessions**: You have the option to open a session in one test case and then, without closing the session, allow other test cases to use the currently open session. Using shared sessions can result in more useful test reports and can save significant time for slow-starting sessions.

If the **Shared sessions** property is blank, then no sessions are shared.

To pass an open session from the parent test case (the test case that includes the **run** step) to a child test case:

a In the parent test case, select the **run** step that starts the child test case.

b If you share sessions, then you cannot enable threaded execution for the step: In the **Step Properties** section, open the **General** properties group. Uncheck **Start this step (in a new thread) and proceed to the next step**.

c In the **Step Properties** section, open the **EXEC run Properties > Run** properties group. For the **Shared sessions** property, specify the **Session ID** (the value in the **Session** cell) of the session that is shared (**session_in_Parent** in our example).

If the session in the child has a different session ID, then use the following syntax to map the two sessions: **session_in_Child=session_in_Parent**

To specify multiple sessions to share with child test cases, separate the sessions using the comma character, for example: **c1=p1, c2=p2, c3=p3, ..., cn=pn**

Now, in the child test case, the **open** step for **session_in_Child** will use the existing **session_in_Parent** session.

Notes A **close** step in a child does not close a shared session — the **close** step in the parent closes the session.

If a specified **Session ID** does not appear in the child test case, iTest ignores the **Shared sessions** setting and uses the session found in the child test case.

Test report for child test cases

While the child test case executes, its output is displayed in the Console view. However, the test case produces its own test report and the resulting execution messages are written into the response of the **run** step. The execution messages do not become a part of the parent test case. The **run** response format is identical to the itestcli format.

The overall test case result (Pass/Fail/Abort/Indeterminate) is used to declare a single Pass/Fail execution issue in the parent test case. The **Indeterminate** result causes a **Fail** declaration on the parent test case.

Tip If a shared session is used by an **open** step in a child test case, then the icon for the step in the report displays a message that identifies the session.

Nesting run steps

You can nest **run** steps. For example, test case A can run several others, including test case B. In turn, test case B can run several others, including test case C (and test case A), and so on.

Executing multiple child test cases concurrently

1 Select all **run** steps.

2 In the **Step Properties** section, open the **General** properties group.

3 Check **Start this step (in a new thread) and proceed to the next step.**

Important If you enable threaded execution for the step, then you cannot use the **Shared sessions** property.

Passing parameter values to the child test case

By default, parameters defined in the parent test case are inherited by and used by the child. You can use this feature to pass values to the child.

By default, parameters defined in the testbed specified for the parent are inherited by and used by the parent.

See “Advanced merging behavior for parameters” on page 755 for instructions on overriding the default merge settings (for example, so that the child ignores the parent parameter values and uses only its own parameters).

Working with the Return result for a run step

The default expected completion result for a child test case is **Pass**. If you do not set pass/fail criteria with an analysis rule, then the completion result is **Indeterminate** and the child test case fails with an **Error** execution issue with an execution message of: **unexpected result "Indeterminate".**

To avoid the problem, add an analysis rule that extracts and analyzes the **TestResult()** query.

In the Response view, right-click the “Indeterminate” part of the **Competition Result** and select **Add Rule > Query > Assert**. To verify that it returns with indeterminate result, then your check_run will now pass.

How topologies or testbeds are used when running child test cases

If a test case calls a procedure in a child test case (a child procedure), then the testbed URI specified in the child test case is not used. Instead, the testbed specified in the calling test case is used.

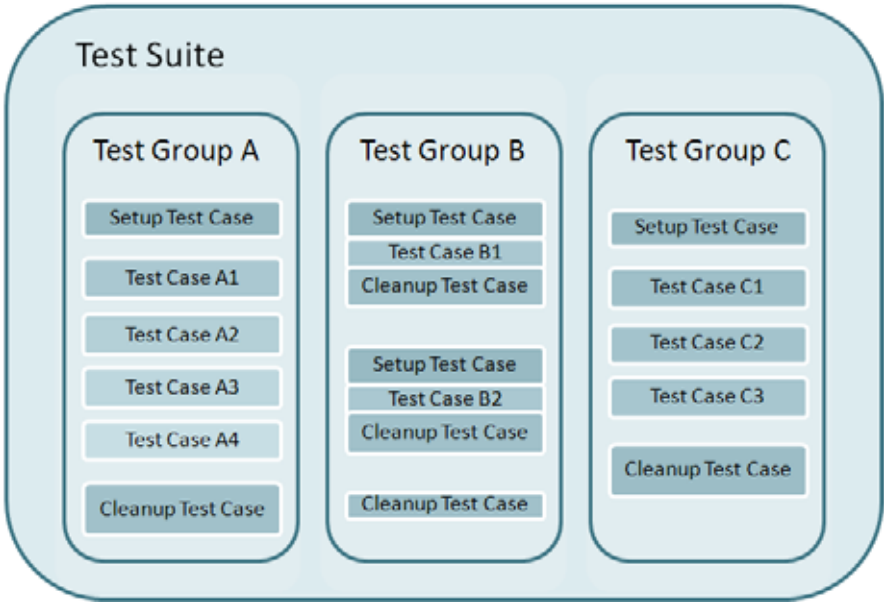
If an **open** step in the child procedure refers to a device URI, then the URI will be replaced using the current testbed that was loaded at start of execution.

Test suites: Organizing tests for group execution

Test suites

Once you define all of the test cases that thoroughly test a device or application, you might want to schedule them for regression testing — all tests to execute one after the other or in parallel. iTest provides the Test Suite editor to simplify the task.

A test suite contains one or more groups of tests. Each test group contains one or more test cases. When you define a test suite, you can specify that each test group should execute in a specified order or concurrently (in parallel). You use the Test Suite editor to specify which test cases should be members of each test group and which test groups make up a test suite. The test suite also specifies the order of execution, timeout limits, and whether setup or cleanup test cases should execute before and after test case execution and/or before and after each test group.



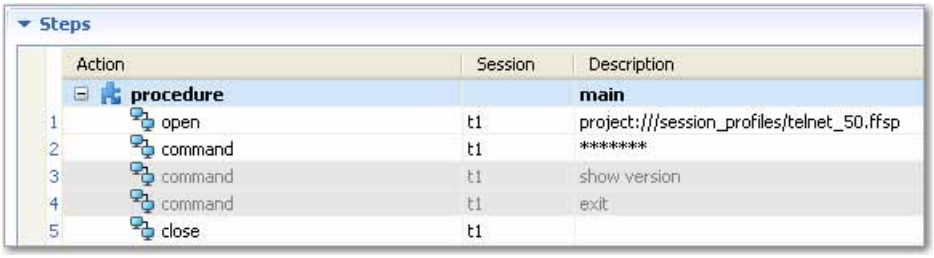
See Chapter 19, “Test Suites”.

Skipping Steps

While developing and troubleshooting a test case, it is often helpful to skip execution for particular steps (like steps that you know work or do not work or that take a long time to complete).

- Skipped steps are not executed and do not appear in reports.
- iTest ignores breakpoints on skipped steps — the steps are skipped without pausing execution

- In the steps grid, skipped steps are dimmed (grayed-out). In the example, steps 3 and 4 are skipped.



Tip Add a **comment** step and indent any number of related steps under the comment. You can then skip all of the steps by skipping only the **comment** step. In addition, you can collapse (fold) the **comment** step to temporarily hide the related steps. See "Tips on working with test case steps" on page 152.

To skip and unskip steps

1 In the Test Case editor, select the steps (use Ctrl+click or Shift+click for multi-select).

2 Click **Skip** .

Click the button again to unskip the step.

Delaying the start of step execution

In some cases, you need to delay the start of a step. For example, the step should wait for a device to reboot or should wait for sufficient debug output when there is no prompt upon completion. In this example, we'll delay the start of a step by 25 seconds:

1 Select the step *before* the step that should be delayed.

2 In the **Step Properties** section, click **Timing > Start**.

3 Set **Normal** to **25**.

4 Set **Fast** to **25**. (The **Fast** setting must be equal to or less than the **Normal** setting.)

Configuring a step to execute in a new thread (asynchronous or concurrent execution)

You can configure a step to run concurrently (asynchronously) with the step or steps that follow it. This feature is commonly referred to as asynchronous execution, asynch execution, concurrent execution, or concurrency.

Note If you use the **Shared sessions** property, then you cannot enable threaded execution for the step. Conversely, if you enable threaded execution for the step, then you cannot share sessions.

1 On the Test Case editor **Steps** page, select the step.

2 Open the **Step Properties** section and click the **General** properties group.

3 Check the box for **Start this step (in a new thread) and proceed to the next step**.

For asynch steps, a marker appears in the first column.

The Threads view enables you to monitor detailed thread execution progress.

For a more complete discussion, see Chapter 36, “Making your test case thread-safe: Actions that help you to synchronize execution threads”.

Setting and accessing variables in test case steps

You can use an iTest variable to hold a single data item or the response data for a test case step (both the text of the response and the structured data part of the response). You can specify multiple values for any variable. iTest supports any kind of variable:

- Numbers
- Strings
- Lists
- Lists of lists
- Arrays [for example, **$a(5)**]

and so on

Naming variables

See “Naming variables and procedures” on page 141.

Local and global variables

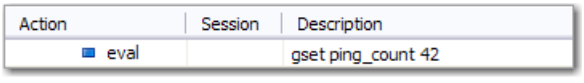
- The values of ***local*** variables are accessible only within the current procedure.
- In contrast, ***global*** variables are available to any step in the test case.

To set variable values

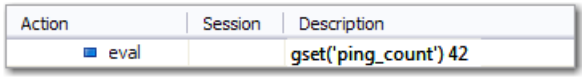
You can create variables in any of the following ways:

- **Set the value directly**

Use an **eval** action with the **set** (local) or **gset** (global) command, for example,



Python



- **Save an entire response into a variable:**

See “Storing a response into a variable (for use later in the test)” on page 140.

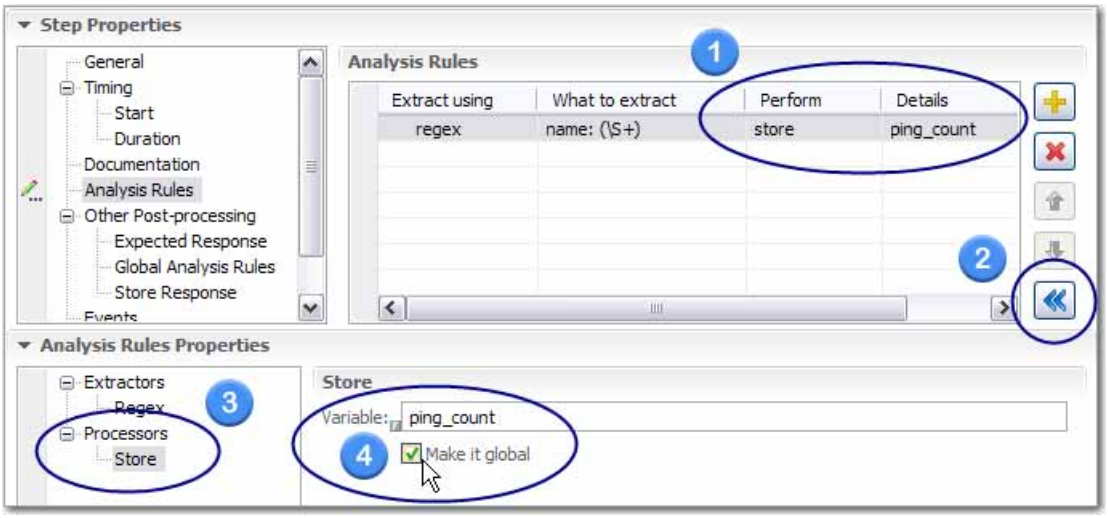
- **Save the result of a query or other extractor in an analysis rule**

1 In an analysis rule **Perform** property, specify **Store**.

2 Click » to access the **Analysis Rule Properties** section. It changes to «.

3 In the **Processors** group, select **Store**.

4 In the step properties, check or uncheck the **Make it global** property as appropriate.



Accessing variable values in a step

Steps can access variable values in either of the following ways:

- Use an **eval** action with the **get** (local) or **gget** (global) command, for example,

Action	Session	Description
eval		ping -c [gget ping_count] 10.155.0.1

Python

Action	Session	Description
eval		ping -c gget('ping_count') 10.155.0.1

- Use an **eval** action with variable syntax:

  ${*varName***}** (Tcl local)

  (‘*varName*’) (Python local)

  ${/data/*varName***}** (Tcl global)

  (‘/data/*varName’)* (Python global)

Action	Session	Description
eval		ping -c ${/data/ping_count} 10.155.0.1

Python

Action	Session	Description
eval		ping -c response('/data/ping_count') 10.55.0.1

- To return data from a variable that holds the response to an earlier step, see “response command: Accessing response data that is stored in a variable” on page 376.

Viewing variables and changing their values during execution

During execution, the Data view displays local variables in the **stack** node in the heap. The **stack** section is transient, that is, it can be **"**popped**"** off and therefore lose all variable information.

In contrast, global variables appear under the **data** node in the heap (instead of the **stack** section). This is what makes the variable global.

See "Data view" on page 277 for information on changing values during execution.

Storing a response into a variable (for use later in the test)

You can set a step property to cause iTest to store the full response for a step into a variable. A subsequent step can make use of the response data by applying a **response** or **query** command to the data in the variable. For example, the step can use the port count value from an earlier response to set the limit for a loop that performs an action on each port.

Options for storing and later accessing the response

You have the following options for storing and accessing the response data:

- By default, the variable is a complex object that includes both the text of the response and the structured data part of the response. To access the contents of the variable (in a subsequent step):
  - Use a **response** command to return all or part of the response's text content. See "response command: Accessing response data that is stored in a variable" on page 376.
  - Use a **query** command to return the result of a query against the structured part of the response. See "query command: Inserting the results of a query" on page 374 and "Applying queries to stored responses" on page 681.
- If you set the **Store only the text of the response** property for a step, then only the response text (and not the structured data part of the response) is stored in the variable. To access the contents of the variable (in a subsequent step):
  - In Tcl test cases, use **$***varName or* **[get** *varName***]** (or, for global variables, **[gget** *varName***]** or **${/data/***varName***}**) to return the full text of the response.
  - In Python test cases, use **'***varName'* (or, for global variables, **gget('***varName')* or **response***('/data/varName/')*) to return the full text of the response.

To store a response into a variable

1. Select the step.
2. In the **Step Properties** section, open the **Other Post-processing > Store Response** property group.
3. In the **Store the response in variable** box, specify the name of a variable that should store the response. (See "Naming variables and procedures" on page 141.)

4 Optional. Check the **Store only the text of the response** check box to save only the response text (and not the structured data part of the response) in a simple variable. As a result, you will later use as follows in Tcl and Python:

Tcl: $*varName or* **[get** *varName***]** (or, for global variables, **[gget** *varName***]** or **${/data/***varName***}**) to return the full text of the response.

Python: *varName or* **get***(varName)* (or, for global variables, **gget***('varName')* or *varName*) to return the full text of the response.

Note Do not check the box if you plan to use a **response** or **query** command on the stored response.

5 Optional. Check the **Make it global** check box to make the specified variable a global variable.

Later, when it is time to read the global variable, use either:

- **[response /data/***varName***]** in Tcl and **response***('/data/varName')* in Python if the entire response was stored. See "response command: Accessing response data that is stored in a variable" on page 376.
- **[gget** *varName***]** or **${/data/***varName***} in Tcl** and **gget***('varName')* or *varName* **in Python** when only the text portion of the response was stored (you checked the **Store only the text of the response** option). See "response command: Accessing response data that is stored in a variable" on page 376.

Naming variables and procedures

iTest uses XML naming rules for variables, procedures, and threads. Please see the formal specification at http://www.w3.org/TR/REC-xml/#sec-common-syn

- Names can contain letters, numbers, and underscore '_'
- Names can start with underscore '_' or any letter, followed by an underscore or by any letter or any number
- Names must not start with a number or any of the remaining punctuation characters
- Names must not start with the letters **xml** (or **XML**, or **Xml**, and so on)
- Names cannot contain spaces
- Names are case-sensitive. **bytes** is different from **Bytes**

Reserved names

The following variable names are reserved by iTest. Do not use the following names:

assertion
index
itest_value
itest_index
procedure
session
value
values

Exchanging data between iTest variables and Tcl variables

Use the **scriptGet** and **scriptSet** commands to exchange data.

iTest supports both Tcl sessions and a default/global Tcl interpreter, so you must specify the Tcl environment/interpreter using the **Session containing interpreter (or specify global Tcl interpreter)** property. (The property appears in the **Step Properties** section, **EXEC scriptGet Properties** and **EXEC scriptGet Properties**.)

- To save an EXEC (eval) value named **evalVar** to a Tcl variable named **tclVar**, use a **scriptSet** step with the following text in the **Description** cell: **tclVar $evalVar**
- To save a Tcl value to an EXEC (eval) variable, use a **scriptSet** step with the following text in the **Description** cell: **evalVar {$tclVar}**

For more detail, see “The ‘scriptGet’ action: Get the value of a variable (Tcl or a selected interpreter)” on page 261 and “The ‘scriptSet’ action: Set the value of a variable (Tcl or a selected interpreter)” on page 262.

Making your test cases more portable

In an open step, if you use the URI of a device (for example, **device:my_devices/DUT_34**) rather than a session profile, your test case becomes portable — it can execute as it is on any testbed that includes a reference to the device. (In contrast, if the step refers to a session profile, then, when you hand it off to a coworker, you have to include the session profile too — the profile can get out-of-synch over time.) Follow this process to create an **open** step that refers to a device URI rather than to a session profile:

1. Start a session from the Favorites view: Open a testbed. Right-click the device and select **Start**.
2. Perform the manual session as you normally would.
3. When you save the captured steps to a test case, the resulting **open** step uses a “**device:**” URI instead of a reference to a session profile. In addition, the **Testbed** property for the test case is updated to refer to the corresponding testbed file.

Notes

- Procedure names are case-sensitive. The procedures named **cleanup** and **Cleanup** are different procedures.
- Variable names are case-sensitive. **$bytes** is different from **$Bytes**.
- Response map names are case-sensitive.

Creating a template for new test cases

If you find that you use a particular set of steps, procedures, and/or property settings whenever you create a new test case, you might consider creating a test case to use as a template. Then, whenever you create a new test case, the steps and properties of the template automatically appear in the new test case.

The template is used only when you create a test case using one of the following methods:

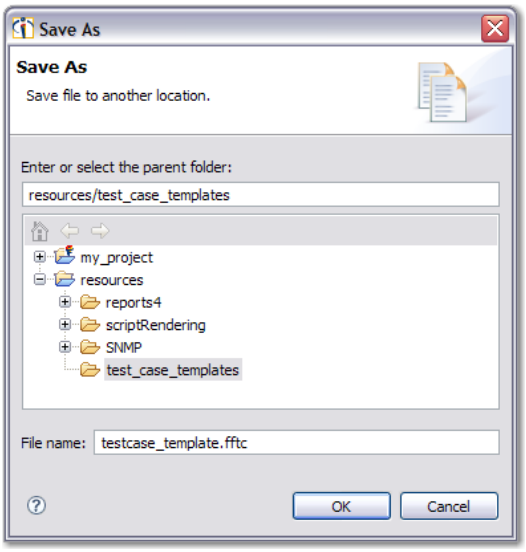
- In the iTest Explorer, right-click the intended parent folder and select **New > Test Case**.

- In the main menu, click **File > New > Test Case**.
- In the main toolbar, click **New Document**  and select **Test Case**.

You can either save an existing test case as the template or create a new test case as the template. In either case, you'll add a test case named **testcase_template.fftc** to the **resources/test_case_templates** folder.

Saving an existing test case as the template

1. While editing the test case, remove or modify items that should not appear in the template.
2. Click **File > Save As**.
3. On the **Save As** dialog box, select **resources/test_case_templates**. (If the **resources/test_case_templates** folder does not exist, create it.)



4. In the **File name** text box, type **testcase_template.fftc**. Click **OK**.
5. The template test case opens in the Test case editor. Edit the template as needed and then save it.

Creating a new template “from scratch”

1. In the iTest Explorer, if the **resources/test_case_templates** folder does not exist, create it: Right-click the **resources** folder and select **New > Folder**. Enter **test_case_templates** into the **Folder name** text box and then click **Finish.**
2. Click **File > New > Test Case**
3. On the **New Test Case** page, click **Browse** and then browse to the **resources/test_case_templates** folder.
4. Specify the **File name** as **testcase_template.fftc** and then click **Finish**
5. The template test case opens in the Test case editor. Edit the template as needed and then save it.

Breakpoints: Overview

Breakpoints tell iTest to pause execution.

In the Test Case editor, when you apply a breakpoint to a selected step by clicking **Toggle Breakpoints** in the toolbar, a circle appears in the margin to the left of the step.

Because breakpoints pause a test case until you manually resume execution, it is important to keep track of them. The Breakpoints view shows you every breakpoint in every test case in your workspace, all at once (even for test cases that are not currently open).

For a more detailed discussion on using breakpoints while developing and debugging test cases, see Chapter 20, “Debugging Test Cases”.

Executing test cases

Executing a test case

For additional details, see Chapter 13, “Executing Tests”. Use one of the following methods:

- In iTest Explorer, right-click the test case and select **Start Execution** , **Start Execution in New Window** , or **Load for Execution**
- In iTest Explorer, select the test case and then click in the main toolbar
- When the Test Case editor is active, click . You may be asked to save the test case before execution.

Scheduling Execution

Scheduling execution

You can schedule tests to execute at a specified time in the future or to execute on a recurring basis once per day at a specified time on specified days of the week. See Chapter 18, “Scheduling Execution”.

CHAPTER 8

Test Case Editor

Overview

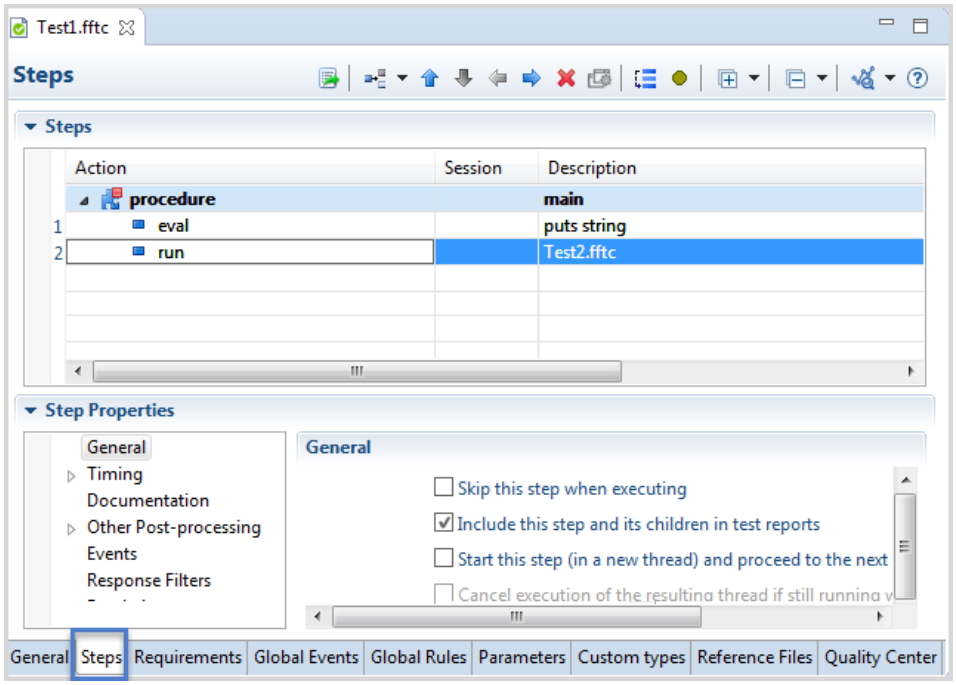
This chapter describes how you use the tools and options available on the Test Case editor pages. For details on the best ways to create, edit, and maintain test cases, see Chapter 7, "Test Cases". In addition, see Chapter 20, "Debugging Test Cases".

Layout of the Test Case editor

A test case describes an automated test to be performed. Use the Test Case editor to edit test case documents.

Each of the tabs along the bottom of the editor provides access to a different page —a collection of information and settings associated with the test case. The **Steps** page is the most important; it lists all of the steps and analysis rules in the test case and shows the flow of execution.

The Test Case editor does more than allow you to configure a test case. It also links the test case to the last associated capture and/or test report and uses the information to update related views (such as the Response, Structure, Queries, and Step Issues views) when you select a particular step. Information on the response for the last execution appears in the views.



Pages on the Test Case editor

Test Case editor: General page See page 162.

Test Case editor: Steps page See page 167.

When you select one or more steps on the **Steps** page, the **Step Properties** section enables you to view and edit property setting for the selected steps. In the example, the **run** step is selected; in the **Step Properties** section, the **General** properties group is selected. Any change to a property setting applies to the **run** step.

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

“Step Properties section: General properties group” on page 176

“Step Properties section: Timing properties group” on page 178

“Step Properties section: Documentation properties group” on page 179

“Step Properties section: Other Postprocessing properties group” on page 180

“Step Properties section: Events properties group” on page 181

“Step Properties section: Response Filters properties group” on page 182

“Step Properties section: Emulation properties group” on page 182

“Step Properties section: Quality Center properties group” on page 183

“Step Properties section: Advanced properties group” on page 183

“Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step” on page 183

Requirements page on the Test Case Editor See page 185.

Test Case editor: Global Events page See page 187.

Test Case editor: Global Rules page See page 187.

Test Case editor: Parameters page See page 187.

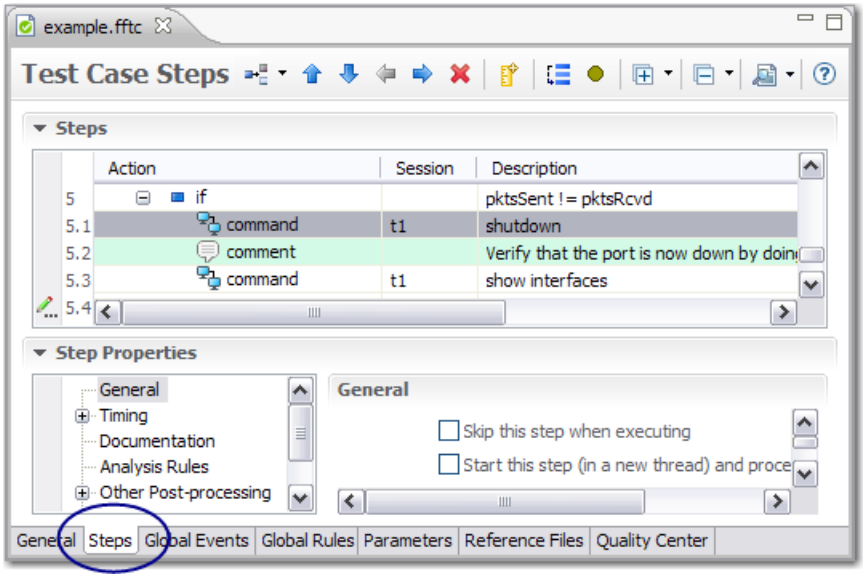
Custom Types page on the Test Case Editor See page 187.

Test Case editor: Reference Files page See page 188.

Test Case editor: Quality Center page See page 189.

Editing test case steps: Basic tools

Use the **Steps** page on the Test Case editor to edit steps. (Click the **Steps** tab to open the **Steps** page.)



- Each step is identified by a unique **Step ID** number in the second column. You can set a preference to display or hide the IDs on the page. Click **Window > Preferences**. In the **iTest** group, go to **Editors > Test Case Editor** and specify the setting.
- When you select one or more steps on the **Steps** page, the **Step Properties** section enables you to view and edit property setting for the selected steps. In the example, the **shutdown** step is selected; in the **Step Properties** section, the **General** properties group is selected. Any changes to the property settings apply to the **shutdown** step. So, to edit a step, select it and modify its properties (in the **Step Properties** section of the page).

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

- To edit multiple steps, select them and change a setting; the setting applies to all selected steps. The same applies to steps that are indented under another step (steps can even be indented under a comment) — any property settings that you make to the parent step apply to the children.)
- Undo/redo (Ctrl-Z, Ctrl-Y) applies to any change, including table operations like moving steps.
- Steps can occur in any number of sessions as long as an **open** step appears as the first step for each session.
- Procedure names are case-sensitive. The procedures **cleanup** and **Cleanup** are different procedures.

- You can copy/paste commands from a text or word file into a test case. The steps are added as commands after the selected steps.
- Test case files use the **.fftc** filename extension

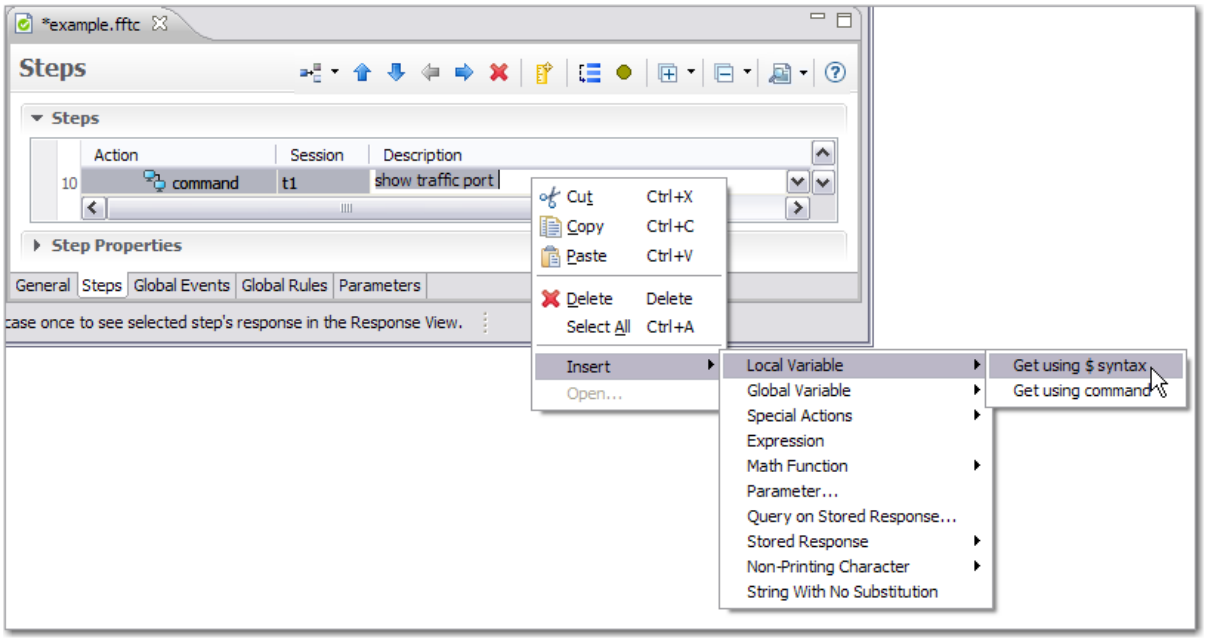
When there is a non-standard property setting or a problem with a step, a marker appears in the first column. See “Markers for steps (step validation)” on page 169.

When in doubt, use the Test Case menu or right-click

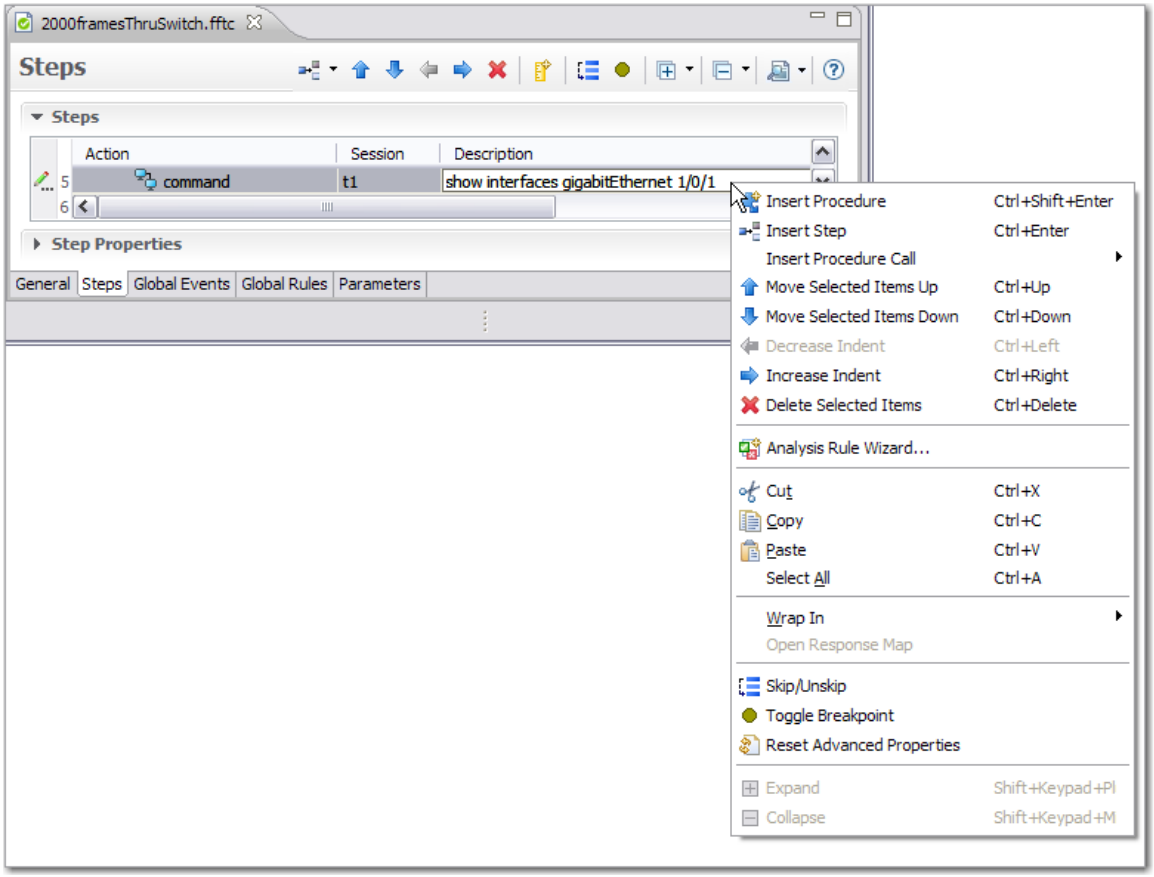
Whenever the Test Case editor is selected, the **Test Case** menu appears in the menu bar. The options that appear in the menu vary depending on the editor page in use.

Most of iTest's editors and views include context (right-click) menus for common operations (most toolbar tools appear in the context menus). For example, in the Test Case editor, there are two menus (details appear later in this topic):

- To insert an item (for example, a variable), select the item to replace or place the cursor at the appropriate location and right-click:



- To modify a step (skip it, wrap it inside a loop or a comment, apply an analysis rule, and so on), select the step and right-click a cell:



Cut/Copy/Paste

If you select a combination of steps and procedures, then only the selected procedures are Cut/Copied/Pasted.

If you select only steps or only procedures, then you can Cut/Copy/Paste them anywhere.

- Steps are inserted after the last selected step
- The steps from a copied procedure are pasted to end of list of steps

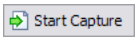
Step: Defined

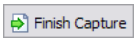
A step is a single row in a test case that often represents a command that you submit to a session. A step can also perform a program control operation like a **for** loop or **comment**, or it can perform an **open** or **close** action to start or end a session.

Action	Session	Description
for		{set i 1} {$i < 2} {incr i}
comment		Insert steps here to be executed on each loop
command	t1	show interfaces gigabitEthernet 1/0/$i
command	t2	exit

Steps are ***executable*** — they are executed when you run the test case and they appear as an executed step in test reports.

Tips on working with test case steps

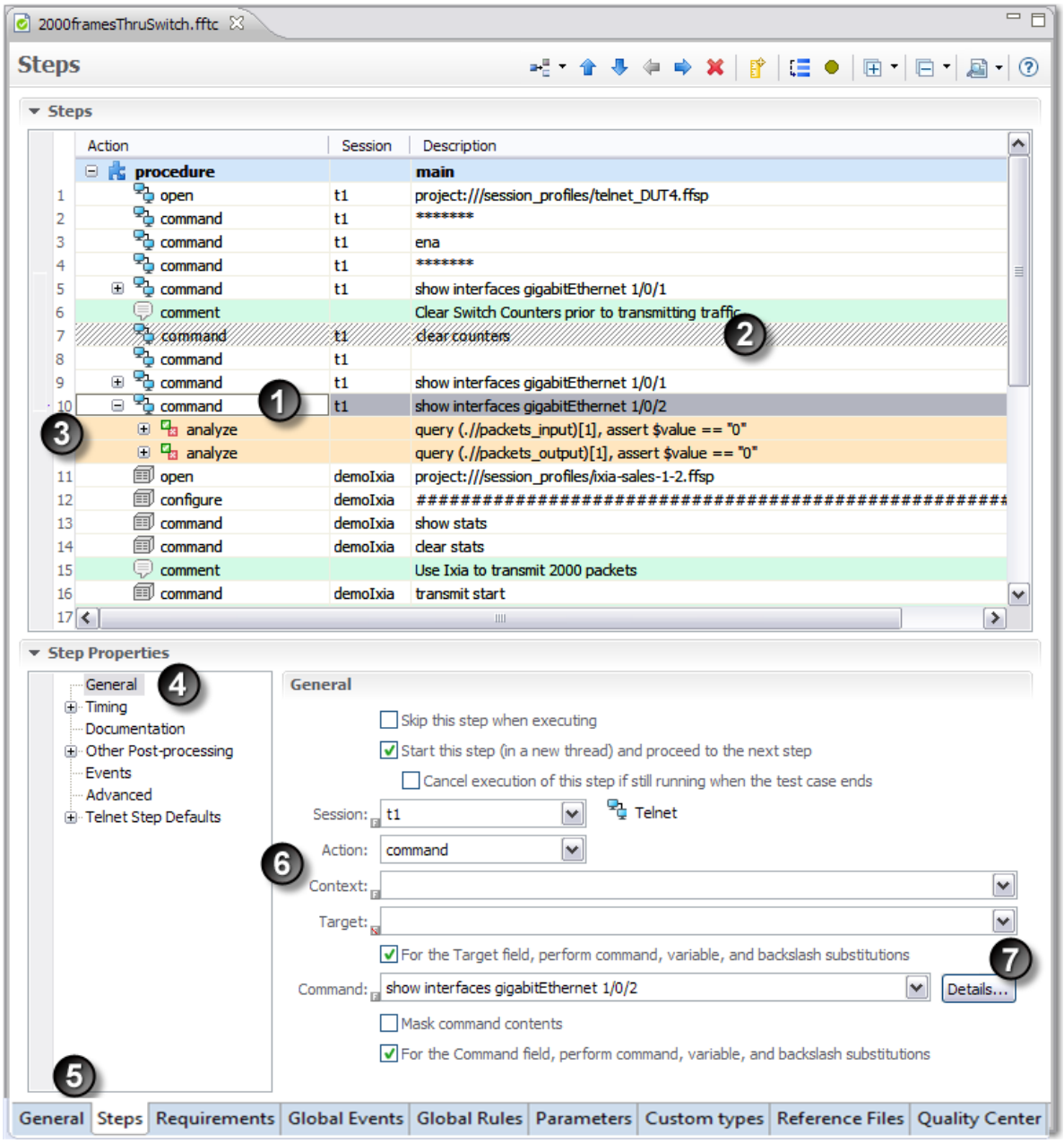
- The fastest way to add steps to an existing test case or to create a test case or procedure is to use the 



buttons to capture directly into the test case. You can also drop captured steps from the Capture view into the Test Case editor or click **Add to Test Case**  in the Capture view. For details on the best ways to create and modify test cases, see Chapter 7, "Test Cases".
- The fastest way to create an **if** statement or a **for**, **foreach**, or **while** loop is to first create the steps, select them and right-click, and then select **Wrap in <loopType>** (or select them and press **Alt-Shift-w**).
- To organize steps and keep a test case looking "clean", you can group associated steps under a **comment** step (the steps execute normally, they are merely indented under the **comment** step to improve readability). After grouping the steps, type appropriate text for the comment into the **Description** cell. As a result, you can collapse or expand a group of steps under the parent **comment** step using  and  . When you collapse all steps, the test case reads like pseudocode so that a coworker can easily understand the operation of the test case. To wrap steps, select them, right-click, and then select **Wrap in Comment** (or select them and press **Alt-Shift-w**).

Working with steps on the Steps page

Most of your time may be spent editing on the **Steps** page of the Test Case editor.



1 Step **10** is selected, so you can:

- Edit its property settings in the **Step Properties** section at the bottom (click the arrow to open the section).

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

- Change the value in the **Action**, **Session**, or **Description** cell in the grid
- Modify it using a toolbar button (described in "Test Case editor toolbar" on page 154) or right-click menu item

2 Selected steps, comments, and procedures are highlighted. Skipped steps (like step 7) are marked with a gray background.

3 If a step has a non-default property setting (like step 10), is breakpointed, or is invalid in some way, then an icon appears in the first column of the **Test Case Steps** grid to identify the issue. Hold the cursor over the icon to view the details.

The circle indicates that step 11 is breakpointed. The icon for step 10 indicates that the step has a non-default property setting (start the step in a new thread — asynch).

4 **Step Properties** section, the step properties are grouped by function. Click **General** to open the **General** section of the Step Properties page.

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select "Show Properties View".

OR

- Click the ellipsis on the step command, where applicable.

---

5 The **Steps** page of the Test Case editor.

6 To specify the operation of a selected step, edit the property settings here.

The **Action** property (both here and in the **Action** cell in the steps grid) is populated with actions that are appropriate for the step's session type.

The **Command** property holds the text of the command.

Required property settings are marked with the * character.

A blue field indicates that setting is being inherited (see the topic on properties inheriting settings).

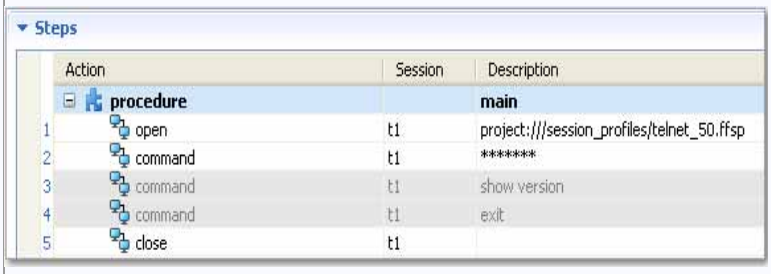
The F indicates that you can use field replacements in the property setting.

7 Click **Details** to define multi-line commands.

Test Case editor toolbar

	**Validate**	This button appears only if you configure manual validation of steps. Click the button to validate all steps. When there is a non-default property setting or a problem with a step, the validation process adds a marker in the first column. By default, iTest auto-validates steps as you create them and the button does not appear. To configure iTest to perform validation only when you request it, see "Properties in: Spirent > Editors > Test Case Editor" on page 190.
	**Insert Procedure**	Add a new local procedure immediately following the selected step.

(Insert)	**Insert Procedure Using Wizard**	Use the wizard to configure a call step and add it. See "Creating a 'call' step using the Procedure Call wizard" on page 229
	Insert Step	Add a new step immediately following the selected step or procedure. **Ctrl-Enter** also inserts a step. iTest uses the combination of **Session name** from the session profile and the unique session number for the day (for example, **myDUT5**) to create the Session ID that appears in the **Session** cell. **Note** By default, iTest sends a carriage return + line-feed sequence when the **Command** cell is blank, so there is no need to include **[char \r\n]** in the **Command** cell for blank commands.
	Insert Analysis Rule	Add a new analysis rule immediately following the selected line. iTest adds a default rule with a regex extractor and assert processor, but you can edit the rule as needed.
	Insert Rule Action	Add a new Action line to the immediately following the selected line in a When True or When False entry for an assert processor in an analysis rule.
	Insert JSON Step	Formulate a JSON document from the test case editor (Provides an easy point-and-click way to perform CRUD operations on nodes in a JSON document) Manipulate the JSON response (provides a programmatic way of performing CRUD operations on **block response** of a procedure). .
	**Move Selected Items Up or Down**	Places the selected steps or procedure immediately before/after the preceding step or procedure. • Steps cannot be moved from one procedure to another. • Steps within looping constructs (if, for, foreach, while) move only within the construct. • If multiple steps are selected, they must be consecutive. • If a procedure is selected then it will be placed before/after the preceding procedure.
	**Increase Indent** **Decrease Indent**	iTest uses indentation to represent membership in loops and other constructs. Move the selected steps down/up one level in the step nesting hierarchy. If multiple steps are selected, they must be consecutive.
	**Delete Selected Items**	Remove the selected steps or procedure from the test case. • Use Ctrl-click and Shift-click to select multiple items • Deleting a **for**, **if**, or **while** construct deletes all steps in the construct • Deleting a procedure deletes all steps in the procedure. **Tip**: **Ctrl-Delete** also deletes the selected items.
	**Analysis Rule Wizard**	Start the Analysis Rule wizard (You must first select the step to apply the rule to.)

	Skip / Unskip	Toggle the skip status of the selected steps, procedures, or analysis rules. (Use Ctrl+click or Shift+click for multi-select.) Skipped steps are marked by being dimmed (grayed -out). In the example, steps 3 and 4 are skipped.  **Tip:** Add a comment step and indent (nest) any number of steps under the comment. You can then skip all of the steps by skipping only the comment step. In addition, you can collapse the comment step to hide the nested steps temporarily. See "Tips on working with test case steps" on page 152.
	Toggle Breakpoint	For the selected steps or procedure, apply or remove a breakpoint. During execution, when iTest encounters a breakpoint, it switches to the **Test Case Debugging** perspective to facilitate single-stepping and other debugging tasks. **Tip:** While execution is paused, you can perform interactive actions in the session to view the results. If a step is useful, you might later add it to the test case. All current breakpoints are identified on the Breakpoints view
	Expand	Click to **Expand All Steps, Summarize Rules** as described here. Click the arrow to select one of the following options: **Expand All Steps, Summarize Rules** (Default): Expand all executable steps and display the first line of any analysis rules **Expand All Steps but Not Rules**: Expand all executable steps and collapse all analysis rules **Expand All Steps and All Rules**: Expand all executable steps and all analysis rules **Expand Selected Items One Level**: For currently selected items, expand the first item in the next nesting level. If the nested item had previously been expanded, then it appears expanded **Expand Selected Items Completely**: For currently selected items, expand all levels of nested items
	Collapse	**Collapse All**: Collapse all executable steps and all analysis rules **Collapse All Selected Items**: For currently selected items, collapse all levels of nested items

	Open recent test reports	While working on a test case in the Test Case editor, the fastest way to view a test report is to click . Click to open the most recent report in the Test Report editor. Click the arrow to display the list of the five most recent reports and then select a report to open it. The Test Report editor is described in "Test Report editor" on page 431.
	Show Properties View	While working on a test case in the Test Case editor, right-click and on the menu displayed, select (**Show Properties View**). The Properties view appears as a tabbed pane on the iTest window. The **Properties View** pane display is context-specific and varies depending on where your cursor is positioned. You may edit step properties using either the **Step Properties** section (within the Test Case Editor) or via the **Properties View** tab. For example, if your cursor is at the **procedure** label (**Steps > Action> procedure**) and then select from the right-click menu, the **Properties** pane opens to display **Procedure Properties** (which is also displayed below the **Steps > Actions** section). **Note** You may position and resize the tabbed Properties view window as required.

Additional in the context (right-click) menu

In addition to all of the tools that appear in the toolbar, the following items appear in the context (right-click) menu:

	Cut / Copy / Paste / Delete / Select All	Typical edit operations
	Wrap In	One of the following: • Wrap the selected steps in an **if** statement, or a **for**, **foreach**, or **while** loop. See "Inserting for, foreach, if, switch, and while constructs into a test case" on page 320. • To organize steps and keep a test case looking "clean", you can group associated steps under a **comment** step (the steps execute normally, they are merely indented under the **comment** step to improve readability). After grouping the steps, type appropriate text for the comment into the **Description** cell. As a result, you can collapse or expand a group of steps under the parent **comment** step using and . When you collapse all steps, the test case reads like pseudocode so that a coworker can easily understand the operation of the test case. To wrap steps, select them, right-click, and then select **Wrap in Comment**. **Keyboard shortcut: Alt-Shift-w**
	Open Response Map	Open the response map associated with the response to the selected step. (The response map is specified in the **Step Properties > Other Postprocessing > Expected Response > Response Map file** property.)
	Reset Advanced Properties	Reset all non-default settings for test case properties, for example, analysis rules, global events, parameter settings, and so on.

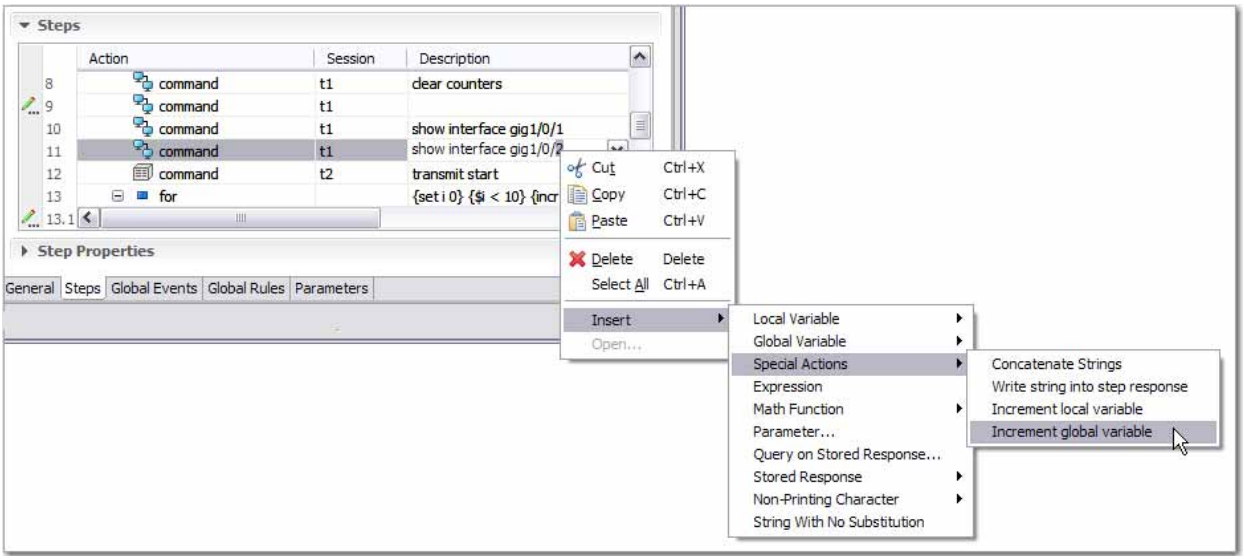
Test Case editor keyboard shortcuts

For a quick list of all keyboard shortcuts click **Help > Key Assist.**

Key	Effect while editing a cell	Effect while a row is selected
Ctrl-Enter (insert step)	• Commits pending changes to the current cell. • For **open** steps, adds a command step below the current row • For all other step types, inserts a new step below the current row with the same Action (and, if appropriate, hint text in the **Description** cell). • Selects the cell in the current column for editing.	For **open** steps, adds a command step below the current row For all other step types, inserts a new step below the current row with the same Action (and, if appropriate, hint text in the **Description** cell). Selects the cell in the current column for editing.
Ctrl-Shift-Enter (insert procedure)	Adds a procedure to the end of the test case.	Adds a procedure to the end of the test case and selects the **Description** cell.
Ctrl-z	Undo	Undo
Tab	Commits changes to the current cell and then moves to the next cell in the same row For the last cell in the row, selects the **Action** cell in the following the row.	Moves the cursor to the next field in the editor, if any. For the last cell in the row, selects the **Action** cell in the following the row.
Shift+tab	Commits changes to the current cell and then moves the cursor to the previous cell in the editor, if any.	Moves the cursor to the previous cell in the editor, if any.
Enter	Confirms the changes to that cell and gets out of the edit mode.	Moves the cursor to the next row, same column.
Esc	Cancels unsaved changes in the current cell and then selects the row (step).	None
F2	None	Selects all text in the current cell and places the cursor at the end of the text. If a single step is selected, puts the cursor in the first editable cell in the step.
F8	None	Sizes each column in the editor to fit the longest cell in that column.

Context (right-click) menu

When you select the contents of a cell, the cell enters edit mode to enable you to change the text as needed. In addition, you can use the context (right-click) menu to make the following selections:



Cut / Copy / Paste / Delete / Select All	Typical edit operations
Insert	Insert one of the items listed in the next table:
Open	The **Open** option appears only when you right-click a session profile in an **open** step. The option opens the session profile in the Session Profile editor (the **New Session** page).

Items in the Insert menu

For further details on the commands mentioned in the table, see Chapter 17, "iTest Commands"

Inserted item	Description and Syntax
Local Variable	Insert an entity that returns the value of a local variable, using either: • **$** syntax of the form **$***varName* • A **get** command of the form **[get** *varName***]**
Global Variable	Insert a field replacement that returns the value of a local variable, using a **gget** command of the form **[gget** *varName***] in Tcl or gget(*'varName'*) in Python**

Special Actions	Use a step with the **eval** Action to perform the following tasks: **Concatenate strings** Concatenates all of the string representations of the arguments into a single string with whitespace between argument strings. [**concat** *string1 string2 ... stringN*] **Write strings into step response** [**puts** *string*] **Increment local variable** [**incr** *varName*] **Increment global variable** [**incr** /data/*varName*]
Expression	Use a step with the **eval** action and the **expr** command to evaluate the specified expression. The result of the query is converted into a string and inserted in place of the command. **[expr** *mathematicalExpression***]** See "expr command: Evaluating expressions" on page 372**.**
Math function	Insert a math function like **abs**, **sqrt**, **round**, **double**, **wide**, and so on. **Note:** iTest displays an error ("Cannot convert to number") if the mathematical expression to evaluate is too big. This is a limitation of the TCL interpreter. Workaround: use variables to save the whole expression step by step. **Example:** set a 29174813892342 set b [math.wide [math.abs [expr 1231-$a]]] In this case b cannot be evaluated. The expression is too large. **Workaround:** Use another variable to save the result of a piece of the math expression: set c [expr 1231 - $a] set b [math.wide [math.abs $c]]
Parameter defined in a testbed or test case	Use a step with the **eval** Action and a **param** command to insert the value of a parameter that was defined in the test case, in the testbed, or in another test case that loaded as a result of a foreign procedure. See "param command: Returning parameter values" on page 372.
Parameter	Open the **Insert Parameter** dialog box to insert a **param** or a **profile** command. See "Inserting a parameter into a property or test case step" on page 741 See also "param command: Returning parameter values" on page 372 See also "profile command: Accessing parameters that are defined in session profile" on page 373 [**param** *parameter_name_or_query ?default_value_if_not_found*?] [**profile** sessionID *parameter_name_or_query ?default_value_if_not_found*?]

Query On Stored Response	Open the **Insert Query On Stored Response** dialog box to insert a query on a response that you had previously stored in a local or global variable. See "Applying queries to stored responses" on page 681. Also see "query command: Inserting the results of a query" on page 374. **How responses are stored into a variable** Responses are stored by setting the **Store response in variable** property for a step. See "Storing a response into a variable (for use later in the test)" on page 140.
Stored Response	The **Stored Response** option inserts a **response** command. The **response** command returns the content of a response that had previously been stored into a local or global variable. For information on using the **response** command, see "response command: Accessing response data that is stored in a variable" on page 376 **How responses are stored into a variable** Responses are stored by setting the **Store response in variable** property for a step. See "Storing a response into a variable (for use later in the test)" on page 140.
Non-printing Character	Use the **char** command to insert a non-printing character (for example, tab, Ctrl-C, Esc, and so on). By default, iTest sends a carriage return + linefeed sequence when the **Command** cell is blank, so there is no need to include **[char \r\n]** in the **Command** field for blank commands. See "char command: Inserting non-printing characters" on page 371
String with no substitution	Insert a fixed string. {*string*}
Information	Insert an **info** command as described in "Commands for returning information: info" on page 358.
File	Insert a **file** command as described in "Commands for managing files and directories" on page 363.

Finding and replacing text in test case steps

On the Test Case editor **Steps** page, select **Edit > Find/Replace** (keyboard shortcut **Ctrl+F)** to search for and replace specific text in the **Action**, **Session**, or **Description** cells.

- You can specify which columns to include or exclude from the find process.
- The find/replace process finds text in any line of multi-line commands. For multi-line commands, the **Command** dialog box opens during the find/replace process, and the process will not continue until the dialog box is closed. **Replace All** does not require that the dialog box opens.
- The replace functionality is undo-able.

Find	Specify the text to find.
Replace With	Specify the text that should replace the text in the **Find** field. The text is replaced when you click **Replace/Find**, **Replace**, or **Replace All**.
Direction	Specify whether to search Forward or Backward from the cursor position. See the **Wrap search** option.

Scope	**All**: Search all properties in all steps in the test case. **Selected Items**: Search all properties in the selected test case steps. **Note** If you select a step that is collapsed (other steps are nested under the step, but are currently folded away), then only the selected step and not the nested steps are searched. To include the nested steps in the search, click **Expand All** before performing the Find process.
Options	**Case-sensitive:** Matches must use the identical case as the text in the **Find** field. **Wrap search:** This option ensures that the entire test case document is searched in the case that you start a search in the "middle". When the find/replace process reaches the end of the document (**Direction** property = **Forward**) or the beginning of the document (**Direction** property = **Backward**), the search continues. **Whole word**: Matches must be the text in the **Find** field surrounded by whitespace. **Regular expression**: Interpret the text in the **Find** field as a regex.
Include Properties	Check an item to perform the find process in the selected column of cells. Each match may include all or a portion of the text in the cell. Matches may not span property (cell) boundaries, with the following exception: The **Description** field is not actually a property but is sometimes a collection of property settings. (For example, for CLI session types, **Description** displays the content of the **Command** property. For Web, Swing, and Flex sessions, it displays the **Target**, followed by the **Command**.) For the find/replace process, **Description** is considered to be a property, so all affected properties are searched/replaced as is appropriate. If a step's **Description** is not editable then the **Description** cell will not be consider for that step.
Find	Find the next instance of the text specified in the **Find** field.
Replace/Find	Replace the currently selected text with the text specified in the **Replace With** field and then select the next instance of the text that you specified in the **Find** field.
Replace	Replace the currently selected text with the text specified in the **Replace With** field.
Replace All	Replace all instances of the text specified in the **Find** field with the text specified in the **Replace With** field. **Replace All** supports multi-line commands.

General page on the Test Case Editor

Test Case editor: General page

General Information

The text that you supply in the **General Information** section appears in:

- The **Summary** section in the structured data for **run** and **summarize** steps
- A column in the Child Test Cases table for responses to **summarize** steps
- Test report **Summary** section in the Test Report editor and in exported test reports.

Headline	Optional. Type a one-line description that documents the usage and function of the test case. In addition to the locations mentioned earlier, this text also appears in the Favorites view to help you when selecting a test case.
Owner	Optional. Type the unique identifier of the person responsible for developing and/or maintaining the test case (typically, the name, login name, or email address), .
Description	Optional. Type additional text that describes the test case to make its usage clear to coworkers. **Tip** This is an excellent place to paste a copy of the test plan.
Test case ID	Information used during test case post-processing
Test case name	Information used during test case post-processing
Namespace	Information used during test case post-processing

Emulation

Enable emulation for the test case	When emulation is enabled, then any step for which emulation is activated will receive an emulated response. See Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions".
Enable emulation duration for the test case	

Test Suite Reporting

This group of properties configures reporting for test cases that are parent test cases that execute child test cases using EXEC.**run** steps. A test case that is made up of only **run** steps is called a test suite.

Because the default settings result in normal reporting for test cases that are not test suites, you do not have to change any of these settings for test cases that do not run other test cases. By default, the response to an EXEC **summarize** step includes the results only of the individual child test cases but not the test suites that executed them. This avoids double-counting of successes or failures.

You can specify the following reporting settings:

Include test cases executed by this test case (via EXEC run)	If this test case runs child test cases (it contains one or more EXEC **run** steps), you might want either to suppress reporting on the results of the children or to include each child's results in EXEC **summarize** steps. Check the box to include the results of child test cases in the response to **summarize** steps. Uncheck the box to not include the results of child test cases in the response to **summarize** steps. Default: Checked
Include this test case in response to summarize steps	This setting applies if this test case will execute as a child test case. Specify whether to or not to display the results of this test case in the response to a **summarize** step in its parent test case. **Yes:** Display the results of this test case in the response to a **summarize** step **No:** Do not display the results of this test case in the response to a **summarize** step **Auto:** • If this test case does not run child test cases (it contains no EXEC **run** steps), then display the results of this test case • If this test case runs child test cases (it contains one or more EXEC **run** steps), then do not display the results of this test case For the **Auto** setting, the response to a **summarize** step will include the results only of children and not of the intermediate parent test cases (this avoids double reporting). Default: Auto
Include execution issues in response to summarize steps	This setting applies if this test case will execute as a child test case. Specify whether or not to display execution messages for this test case in the response to a **summarize** step in its parent test case. **Yes:** Display execution messages for this test case in the response to a **summarize** step. **No:** Do not display execution messages for this test case in the responses to a **summarize** step. **Auto**: • If this test case does not execute child test cases (it contains no EXEC **run** steps), then include this test case's execution messages • If this test case has one or more child test cases (it contains one or more EXEC **run** steps), then do not include this test case's execution messages. For the **Auto** setting, the response to a **summarize** step will show only a list of the child tests and their results, and not the intermediate parent test cases and results (this avoids double reporting). Default: Auto

Local Topology

Local topology or testbed	Optional. Specify a topology or testbed to use for the sessions in the test case. Testbeds define devices and specify a list of session profiles and can set parameter values. Topologies define devices and specify a list of session profiles. When you specify a topology, you can make the test case more flexible by using parameters defined in the associated session profiles to customize behavior at runtime. If the **Local topology or testbed** property is blank or no Global topology or Global testbed is specified, then **device** URIs are not supported in **open** steps. **About Global topology or Global testbed** If you have identified a Global topology or testbed and you also specify a Local topology or testbed, then, at runtime, you will be asked whether to use the Global topology or testbed or the Local topology or testbed during execution. You can configure whether the dialog box should or should not appear or that either the Local or Global topology or testbed should be used when this condition arises. See "Setting preferences for execution" on page 281. Also, see "Global topology" on page 539 and "Global testbed" on page 94. **Test documentation includes the topology or testbed** When a topology or testbed is used for execution, to help you keep track of which topology or testbed was used, an informational execution message will appear immediately after the execution start message. The message identifies the fully qualified URI of the topology or testbed that is being used. In addition, the URI appears in the header section of the test report.

Library Settings

Procedure libraries	**Note** You have the option to include the name of this test case in the drop-down list of procedure libraries while creating a call step. Check the box to include the name of this test case in the drop-down list of procedure libraries while creating a **call** step. When you check the box to identify this test case as a procedure library that should be made easily available when creating test cases. As a result, when you add a **call** step or a **CallProcedure** action to a test case, the URI for this test case appears in the drop-down list of procedure libraries and local and foreign procedures in the **Description** cell. When you uncheck the box, the URI does not appear in the drop-down list. Default: unchecked **Note** Click the link to open the **Test Case Steps** page so that you can edit steps and procedures. Procedures are described more full in Chapter 10, "Procedures".
QuickCall Library	The settings in this section identify the test case as a QuickCall Library, make the library "public", and associate the QuickCalls with a testbed device or session profile For more information on QuickCalls, see Chapter 9, "QuickCalls: Defining and using a library of custom actions". Check box to make the library "public", that is, to cause iTest to display the test case name whenever a user asks to see a list of available QuickCall libraries. This settings adds the QuickCall library to the drop-down list when you edit a session profile or device to associate the library.
Associated Session profile	Availalbel only when you select QuickCall Library. Specify the testbed device or session profile to associate with the QuickCalls that are defined in the library. Once you have specified a device or profile, the link becomes active and opens the item in the appropriate editor.

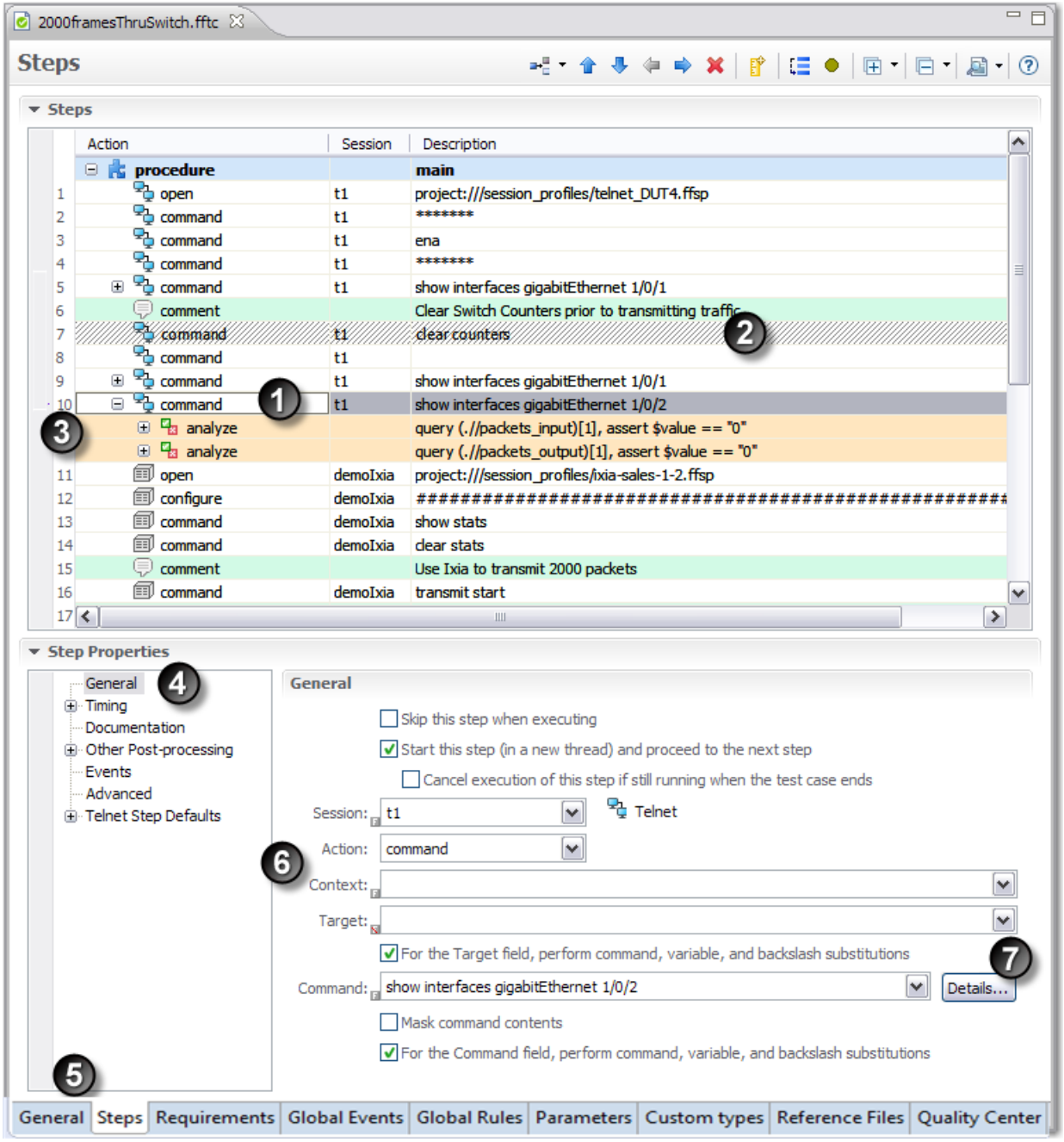
Execution Behavior

Language	**Language:** Select the language that will be used to create the session profile. You may use the default language displayed (as set in "Preferences: Spirent > General > General preference settings" on page 863, Chapter , "Configuring iTest Preferences") or select a different language from the list.
Entry point	The first procedure to execute when you execute the test case. Typically, **main**. All procedures defined in the test case are listed.
Generate an execution message for each comment step that is executed	Check the box to cause steps with an **Action** of **comment** to display execution messages in both the Execution view and test report. See "The 'comment' action: Add a comment to a test case" on page 246. Default: unchecked
Force all variables to be global	**Warning** This option is not recommended for new iTest test cases. It is intended for use with imported FanfareSVT test cases to ensure proper operation (all FanfareSVT variables are global). Check the box to cause all variables (local or global) to be treated as global. **Warning** When using this option, take care with procedure arguments, especially unnamed procedure arguments, as these will all appear in the same global namespace and can therefore cause problems with nested procedure calls. Default: unchecked
Discard session profile parameters when sessions close	Check the box to ensure that parameter values are removed from memory when the current session finishes. **We strongly recommend** that you leave the box checked. Default: checked
Execution time limit	Specify the time limit for the overall test case. iTest stops the test case if it exceeds the limit. **hh:mm:ss** format
Default step timeout	Specify the maximum time limit for any step in the test case. iTest stops the test case if it exceeds the limit. **hh:mm:ss** format
Estimated execution time	This value represents iTest's best estimate of how long it will take to execute the test case. The value is updated after each uninterrupted execution. The value is not updated when the test case is executed by an EXEC **run** step or a test suite. The value is used in two ways: • On the Execution view during execution, to determine the progress on the **Time remaining** progress bar • For use by test case scheduling applications You have the option to edit this value as needed. For a test case that has never been executed, the default value is 00:00:00.
Update "Estimated execution time" value after execution	Check the box to cause iTest to update the estimate when execution ends. The algorithm is (0.7*<current value>) + (0.3*<duration of latest execution>). The **Estimated execution time** value is not modified if: • The user cancels execution • The test case is executed by an EXEC **run** step or a test suite • Test result is **Fail** or **Abort** • Execution speed setting is not "fastest possible" Default: Checked

Steps page on the Test Case Editor

Test Case editor: Steps page

On the **Steps** page, you review, add, and modify steps.



1 Step 10 is selected, so you can:

- Edit its property settings in the **Step Properties** section at the bottom (**#4** — click the arrow to open the section)

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

- Change the value in the **Action**, **Session**, or **Description** cell in the grid
- Modify it using a toolbar button (described in “Test Case editor toolbar” on page 154) or right-click menu item

2 Selected steps (like step **10**), comments, and procedures are highlighted. Skipped steps (like step **7**) are marked with a hashed background.

3 If a step has a non-default property setting (like step 10), is breakpointed, or is invalid in some way, then an icon appears in the first column of the **Steps** grid to identify the issue. Hold the cursor over the icon to view the details.

The circle indicates that step 11 is breakpointed. The icon for step 10 indicates that the step has a non-default property setting (start the step in a new thread — asynch).

4 Down here in the **Step Properties** section, properties for the step are grouped by function. We clicked **General** to open the **General** properties page (to the right).

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

5 We're viewing the **Steps** page of the Test Case editor.

6 To specify the operation of a selected step, edit the property settings here.

The **Action** property (both here and in the **Action** cell in the steps grid) is populated with actions that are appropriate for the step's session type.

The **Command** property holds the text of the command.

Required property settings are marked with the * character.

A blue field indicates that setting is being inherited (see the topic on properties inheriting settings).

The F indicates that you can use field replacements in the property setting.

7 Click **Details** to define multi-line commands.

Tips on working with test case steps

- The fastest way to add steps to an existing test case or to create a test case or procedure is to use the Start Capture Finish Capture buttons to capture directly into the test case. You can also drop captured steps from the Capture view into the Test Case editor or click **Add to Test Case** in the Capture view.

- The fastest way to create an **if** statement or a **for**, **forEach**, or **while** loop is to first create the steps, select them and right-click, and then select **Wrap in <loopType>** (or select them and press **Alt-Shift-w**).
- To organize steps and keep a test case looking “clean”, you can group associated steps under a **comment** step (the steps execute normally, they are merely indented under the **comment** step to improve readability). After grouping the steps, type appropriate text for the comment into the **Description** cell. As a result, you can collapse or expand a group of steps under the parent **comment** step using ⊞ and ⊟. When you collapse all steps, the test case reads like pseudocode so that a coworker can easily understand the operation of the test case. To wrap steps, select them, right-click, and then select **Wrap in Comment** (or select them and press **Alt-Shift-w**).

Markers for steps (step validation)

By default, iTest auto-validates steps as you create them. When there is a non-default property setting or a problem with a step, a marker appears in the first column. Hold the cursor over the marker to read the details or double-click it to view the setting in the **Step Properties** section of the editor.

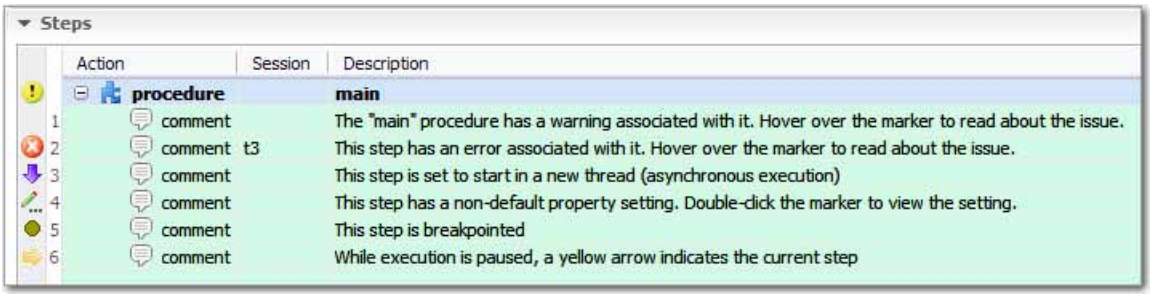
Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

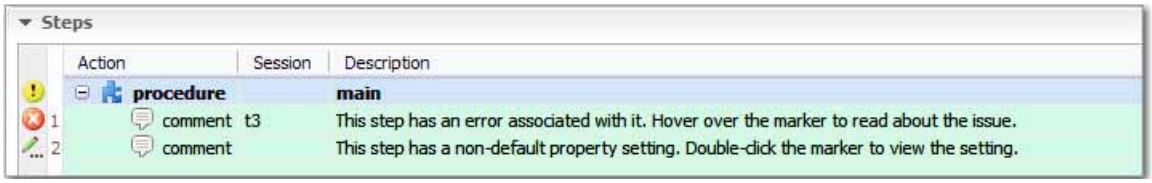
Note If there is an issue with an **open** step, then a marker appears for the **open** step, but no warning or error markers appear for the other steps in the procedure (in the current workspace). In other procedures that include a step in the problematic session, a marker appears for the first step in the session. When the issue with the **open** step is resolved, other markers may appear (for other reasons) for steps in the session.



Controlling validation of steps and property settings

By default, iTest auto-validates property values as you set them. Validation determines whether there is a a problem with a step and whether any property settings are invalid or non-default.

- In the Test Case editor, the validation process marks a step with an icon in the first column:



- In the Session Profile editor, the validation process marks a problematic property value with an error marker and identifies a non-default property setting by changing the property value field from blue (default) to white (non-default).

You have the option to configure iTest to ***not*** perform validation — steps are not auto-validated and no markers appear for invalid or non-default property settings.

- If auto-validation is disabled, you can perform validation on-demand in the Test Case editor — click **Validate**
- You cannot perform on-demand validation in the Session Profile editor.

You control the option using the **Perform step validation only when requested** property on the **Spirent > Editors > Test Case Editor** preferences page. See "Properties in: Spirent > Editors > Test Case Editor" on page 190.

Action

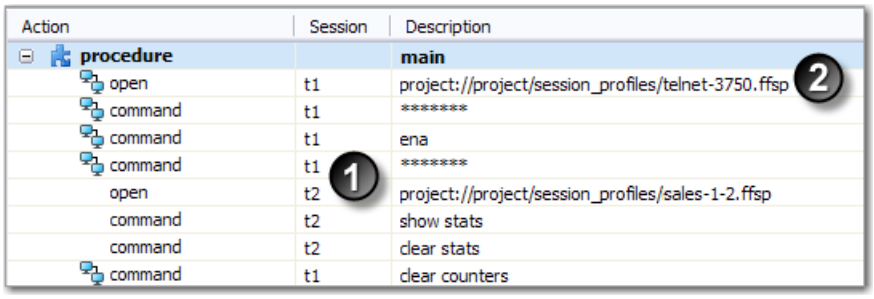
While adding a step in the Test Case editor, you use the **Action** cell to specify the action for the step (for example, perform a CLI **command**, add a **comment**, or do an SNMP **get**).

See "Actions" on page 239.

Session

The **Session** is a unique identifier for the session that is associated with the step. A test case can open any number of sessions. The first action for each session must be an **open** action.

iTest typically auto-assigns the **Session** ID for a session in the **open** step for the session. All steps in the session must specify the value in the **Session** cell.



For devices with more than one session attached or for multiple captures that use the same session profile: To create the Session IDs that appear in the **Session** cells in the Test Case editor and in test reports, Spirent iTest uses the combination of **Session name** from the session profile (for example, **myDUT**) and a unique session number for the day. (for example, **myDUT.1** and

myDUT.2). If the session profile does not specify a **Session name**, then Spirent iTest uses the filename of the session profile in its place.

❶ In this example, we named the two sessions **t1** and **t2** (we used **t** to represent Telnet). You can use more descriptive names if you like.

Tip When you are ready to create dynamic QuickCalls or procedures, you can pass the **Session ID** as a variable. For example, use **$my_session** in the **Session** cell rather than the hard-coded value **t1**.

---

❷ This is the URI of the session profile that the **open** step uses to start session **t1**.

The session profile specifies the session type, the DUT to connect to, and the configuration of the session window (the terminal for CLI session types or the browser for graphical interfaces).

Description

The value, arguments, or command text that implements the action (for example, a comment, a session profile URI, or command text).

You can edit the text directly. The text that appears in the **Description** cell reflects the property settings that you make in the **Step Properties** section.

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

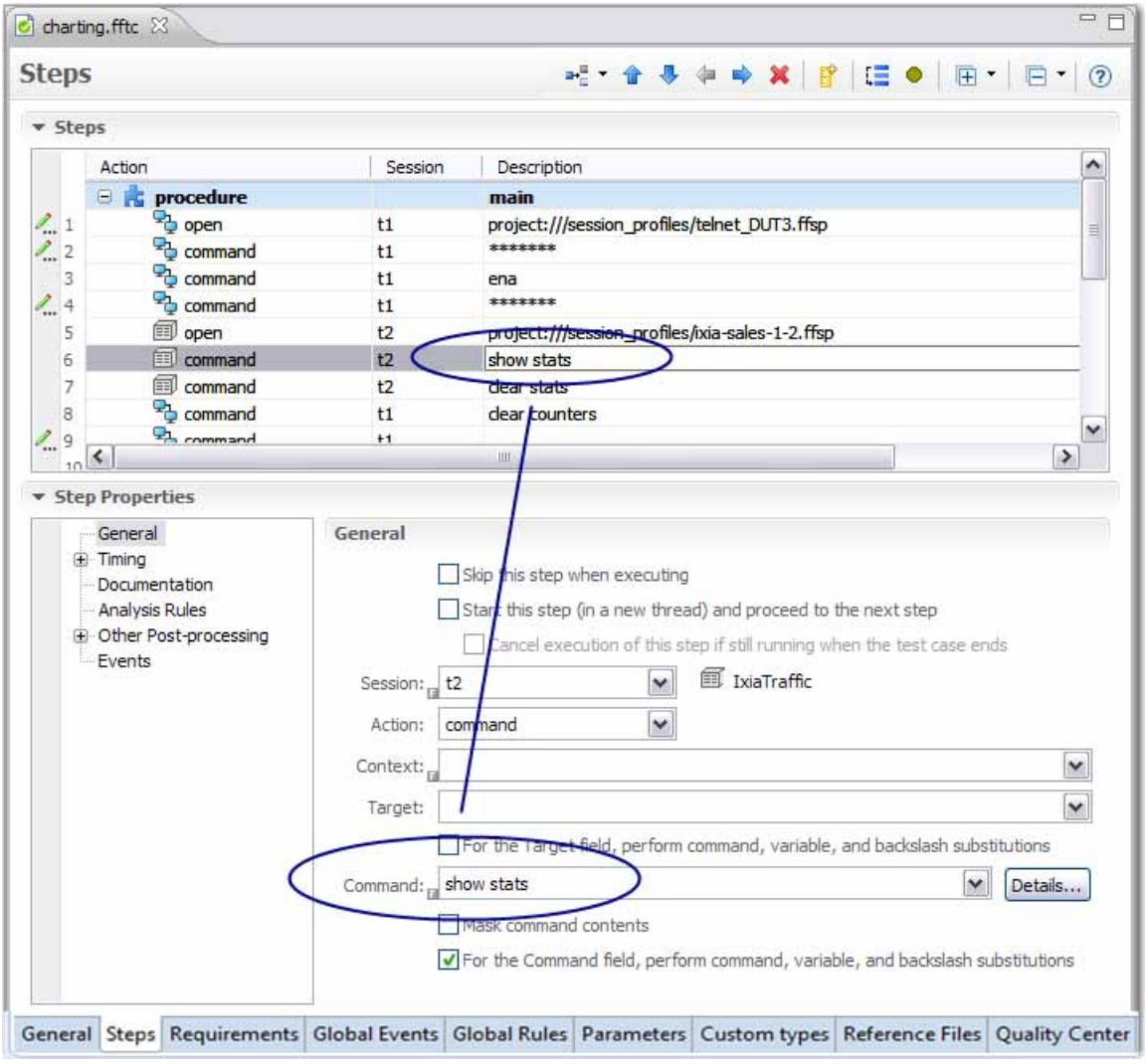
- Right-click to display the menu and select “Show Properties View”.

OR

- Click the ellipsis on the step command, where applicable.

For most other session types, The **Description** cell displays the **Command** property value. Web, Swing, and Flex steps differ from steps for other session types. In steps for browser-based sessions, the **Description** cell displays the following values, separated by colon and semicolon: **Context:Target; Command**

Properties can inherit their settings. See “Property values: Inheriting settings” on page 62.


charting.fftc
Steps
Steps
Action
Session
Description
procedure
main
1 open t1 project:///session_profiles/telnet_DUT3.ffsp
2 command t1 *******
3 command t1 ena
4 command t1 *******
5 open t2 project:///session_profiles/ixia-sales-1-2.ffsp
6 command t2 show stats
7 command t2 clear stats
8 command t1 clear counters
9 command t1
Step Properties
General
Timing
Documentation
Analysis Rules
Other Post-processing
Events
General
Skip this step when executing
Start this step (in a new thread) and proceed to the next step
Cancel execution of this step if still running when the test case ends
Session: t2
IxiaTraffic
Action: command
Context:
Target:
For the Target field, perform command, variable, and backslash substitutions
Command: show stats
Details...
Mask command contents
For the Command field, perform command, variable, and backslash substitutions
General
Steps
Requirements
Global Events
Global Rules
Parameters
Custom types
Reference Files
Quality Center

Step Properties (re-direct to properties configuration page)

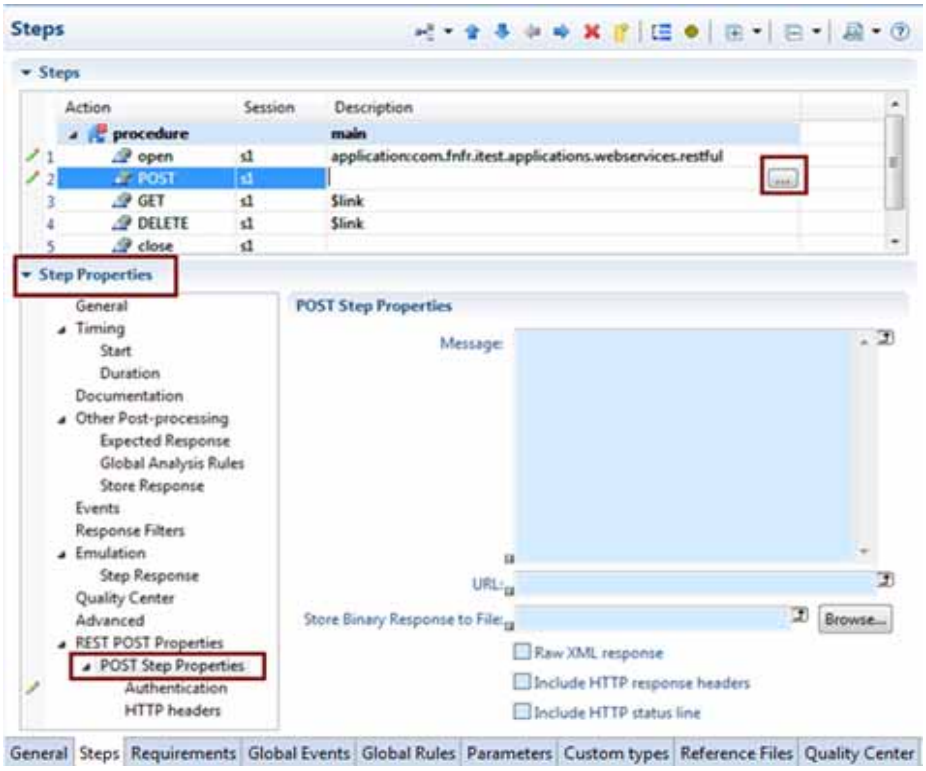
For ease of locating and changing property of some steps, click **…** is appended to the **Description** cell of some of the steps to easily navigate to the corresponding step property page. For example, see “POST” step below. Clicking **...** expands the **Step Properties** section, selects **POST Step Properties** node on the property tree and also expands its children nodes.

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select “Show Properties View”.

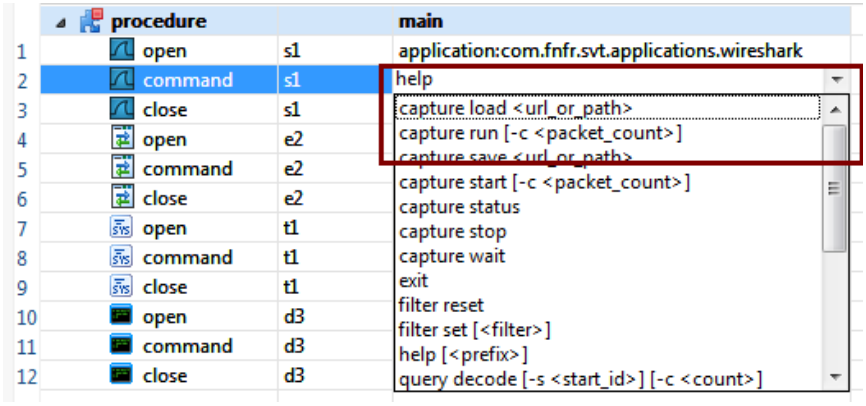
OR

- Click the ellipsis on the step command, where applicable.



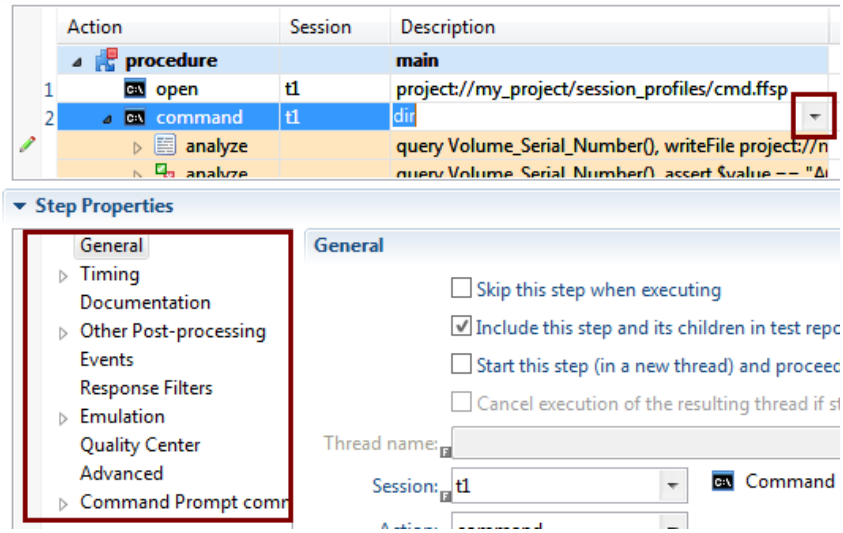
Being re-directed to properties configuration page (clicking ...) applies to all test case steps with the below exceptions.

1 If an existing test case step has a non-empty combo box, the … button is not available.

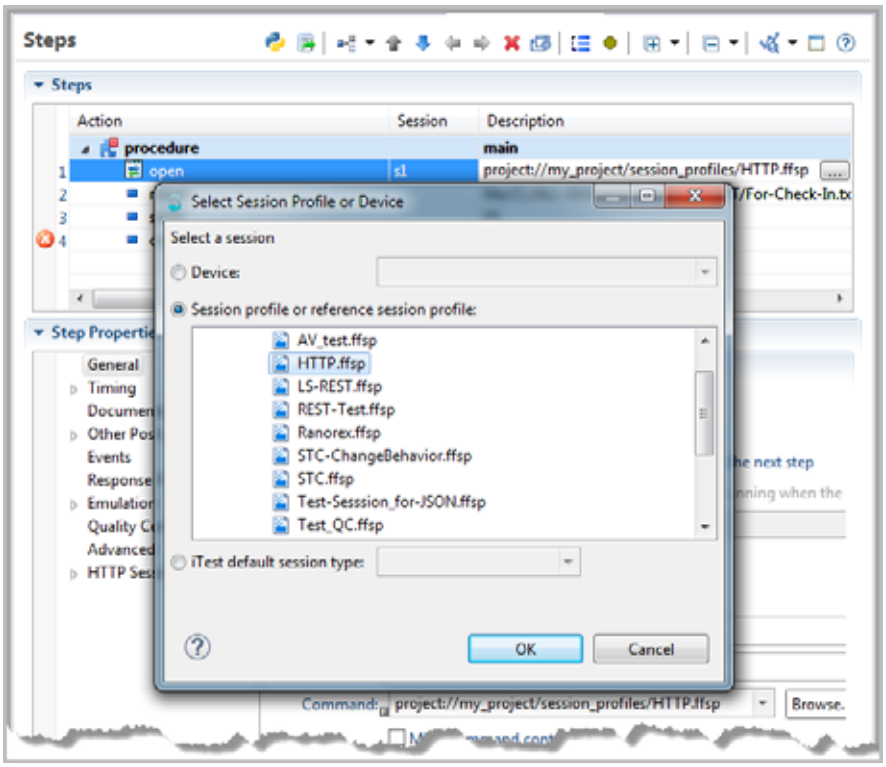


2 If steps have no “Xxx Step Properties” node on the property tree, iTest does not display the ... button (see the drop-down option available).

Note “Xxx” is the action name of the step and the first letter is not case-sensitive.

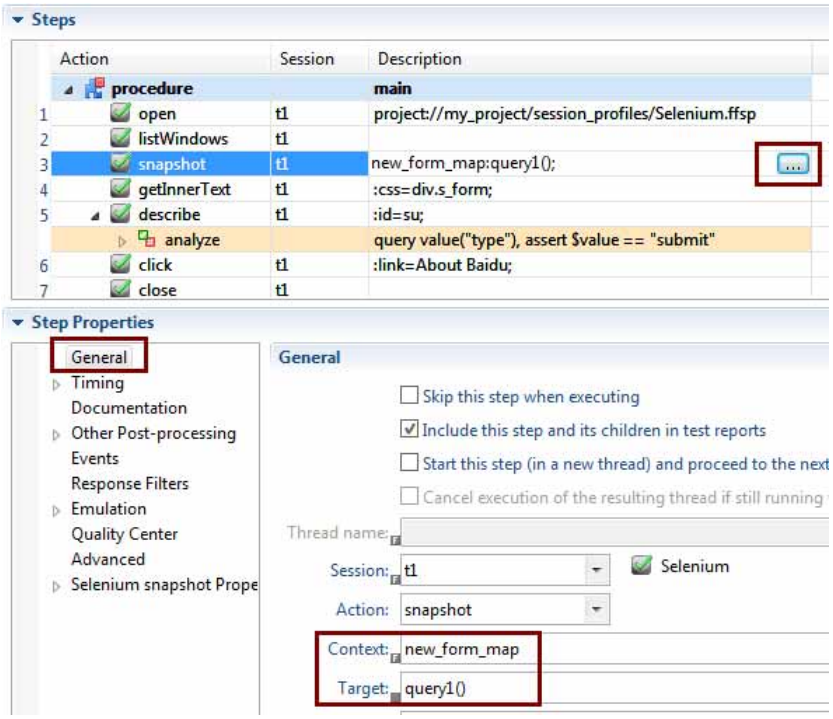


3 For some steps (e.g., open step), the **...** (elipsiss) button displays a dialog or wizard rather than step properties.



4 For all session types that involve GUI operation (e.g., Selenium, web, Flex, Swing and Ranorex), that is, for those session where the “Target” on “General” page is meaningful, if the form map is specified in the corresponding session profile, whether the current step

("close" step is excluded) uses form map or not, clicking **...** selects the Step Properties > **General** node to show **Context** and **Target** by default.



Step Properties section: General properties group

Note You may open the context specific information—**Step Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select "Show Properties View".

OR

- Click the ellipsis on the step command, where applicable.

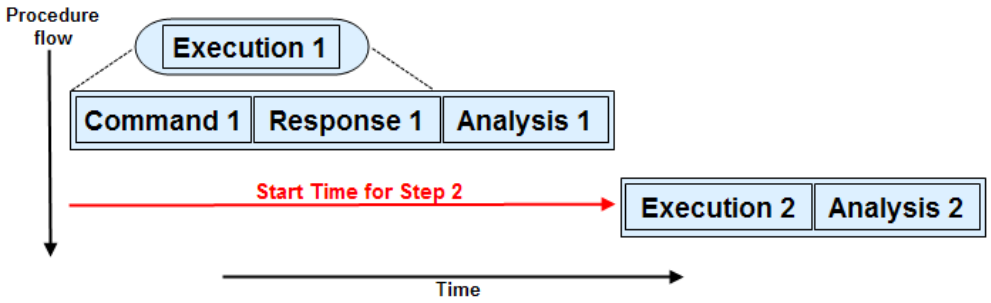
General properties group

Skip this step when executing	Prevent the step from being run when the procedure is executed. Skipped steps have a gray dotted background in the Test Case editor.
Include this step and its children in test reports	By default, each step that executes in a test is added to the test report. Uncheck the box to specify that a particular step and any of its children should not appear in test reports (the step executes normally, it simply is not reported). **Execution issues** If a child step of any step that is configured not to appear in reports has an execution issue, then, in the in the list of execution messages in the Execution view, the issue's icon appears next to the message for the nearest ancestor. The step index for the issue is associated with the child step that had the issue. **Overriding this setting** You can override this setting for all test cases by setting a preference. See "Controlling which executed steps appear in test reports" on page 439. Default: checked
Start this step (in a new thread) and proceed to the next step	Cause the step to run after the preceding step and concurrently (asynchronously) with the following step or steps. This feature is commonly referred to as Specify how long to wait for the prompt to appear execution, async execution, concurrent execution, or concurrency. See "Threads view" on page 276
Cancel execution of the resulting thread if still running when the test case ends	Some steps may still be waiting for a response when the test case ends. Check the box to end the test in this case. The test result is set to **Fail**.
Thread name	Specify a name for the thread associated with this step. See Chapter 36, "Making your test case thread-safe: Actions that help you to synchronize execution threads".
Session	The reference label for the session associated with the step. A procedure may access any number of sessions, with the first action for each being the **open** action.
Action	The session-specific action to take for the step. See "Actions" on page 239
Command	Typically, the **Command** property holds the command to send to the session as part of the associated **Action**. Additionally, the property can hold the URI of a file. **Examples** • For a CLI **command** action, the **Command** might be **show routes**. • For an **open** step, the **Command** might be the URI of the session profile or the name of the device to use to start the session • For a **comment** step, the **Command** is the text of the comment. • For a **configure** step (used by many traffic-generator device session types), the **Command** is the text of the multi-line configuration file For **open** and **readFile** steps, **Browse** opens a dialog box so you can specify the file to access. For steps that involve text, **Details** opens a multi-line text box (so you can enter, for example, a multi-line command). Type or paste the text into a text box.

Mask command contents	Check this box to cause iTest to display sensitive information (passwords, for example) as asterisks.
For the Command field, perform command, variable, and backslash substitutions	Check the box if the string specified for the **Command** property uses a command field replacement, a variable, or a backslash that is used to escape a special character. As a result, the substitutions will be performed before iTest uses the text that appears in the **Command** field.
Command field contains an encrypted value	Check the box if the **Command** text includes a reference to a parameter whose value is masked (encrypted and hidden from the user). For example, the text might include a **param** command in a field replacement. For instructions on masking a parameter's value, see "Working with parameters: The Parameters page" on page 733.

Step Properties section: Timing properties group

Unless steps are set to run asynchronously (concurrently), a step starts when the previous step has completed. The beginning of each step is timed relative to the beginning of the previous step, rather than absolutely.



Timing > Start

Adjust the properties on this page to change when a step begins relative to the beginning of the previous step.

Normal and **Fast** set the **"**scale**"** on the speed control in the Replay view. **"**Original**"** in that control is defined by the **Normal** property. The "Fast" value on the speed control is set by the **Fast** property. **"**Slow**"** is 10 times the **Normal** property.

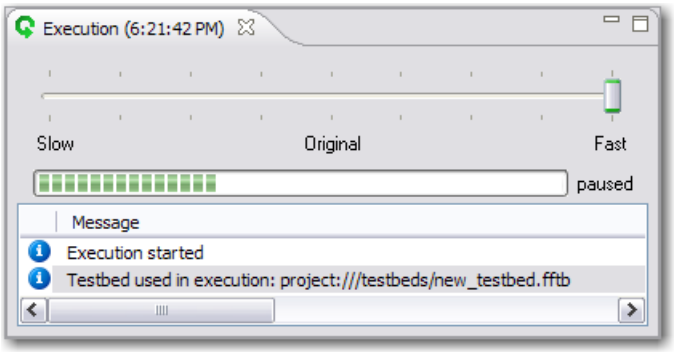
Remember that all of these offsets are **minimum** delays for starting the selected step after the **beginning** of the previous step. Unless the previous step is set to run asynchronously iTest always waits until the end of that step before beginning the next step. If the preceding step takes longer than the delays set here then the step will execute immediately upon completion of the previous step.

If you set **Normal** to 10 seconds and then select **Original** as the execution speed, then the step will start 10 seconds after the previous step starts even if that step takes one second to execute.

You can set **Fast** to be less than **Normal**. If you set **Fast** to 0 seconds and then select **Fast** as the execution speed, then the step will start as soon as the previous step ends.

- During execution, change the execution speed at any time.

- Specify the default speed for any test case step in the **Timing > Start** properties group.



Note Certain steps require a minimum time to execute. For example, if you send four ping packets to a server, then the **ping** command is not complete until all four packets have been sent. In such a case set the **Fast** property for that step to allow the command enough time to execute properly even on the fastest replay speed.

Timing > Duration

Use the default step timeout setting specified for this test case	Check the box to override the **Default step timeout** that is specified on the **General** page. Click the link to move to the **General** page and view the current setting that is applied to all steps in the test case. Default: checked
Timeout for this step	If the **Use the default step timeout** checkbox is unchecked, then specify the timeout value for this particular step, in seconds.
Duration	Expected duration of the step in seconds. This value is auto-populated after each execution and supports the Execution Progress bar in the Execution view. This is only a note, and has no effect on step execution.

Step Properties section: Documentation properties group

The information on this page does not appear during execution. It serves as an internal documentation and sorting tool and provides backward compatibility with features of FanfareSVT

Documentation properties

Label	**Note** Intended only to support imported FanfareSVT **EXEC.goto** steps. The **goto** action is deprecated in iTest. When you specify text for the label, then a **goto** step can refer to this step in the **Description** cell. **goto** steps work only within procedures.
Tag	A tag is user-defined text string that provides backward compatibility with FanfareSVT. When you import a FanfareSVT test case, the **Tag** property values are placed into this field. The **Tag** property is used for documentation only.
Comment	Type a comment string. You might provide more information about the step to make it easier to understand what is happening. For example, “Enter Interface Configuration mode” or “Reset Ethernet settings”. In addition, you could use the Comment to remind other automation engineers about the required context for the step. This Comment differs from the **Comment** action, which outputs a user-defined string into the capture report.

Step Properties section: Other Postprocessing properties group

Expected Response

Do not use a response map for this step	Do not use a response map for the step. Use this setting when the response might match a response map due to the map's **Applicability** setting and you do not want the response to map **Note** Even if you select this option, iTest will apply an auto-generated response map if you check the **Use an auto-generated response map if no other map is available** property.
Use the response map library configured for the session	Use the response map library specified in the session profile associated with the step. iTest searches the library and applies the first map with appropriate **Applicability** settings.
Use a response map file	Use the response map specified for the **Response Map file** property. If you check **Use a response map file**, then specify the file in the **Response Map file** text box. Once you specify a response map, the **Response Map file** label becomes a link. Click the link to edit or review the response map.

Find the response map by name in response map library configured for the session	**Response map name** When you select this option, iTest searches the response map library associated with the step's session for the specified response map that you specify in the **Response map name** text box. iTest applies the first map with the correct name and with appropriate **Applicability** settings. If the response map is not found, then an **OnResponseMapNotFound** event occurs. **Response map to use at design time** **Note** If you do not use variables or field substitution in the **Response map name** value, then you cannot specify a value for **Response map to use at design time** If you use field replacement in the **Response map name** setting to determine the response map to use at runtime, then, at design time (while you are working in the Test Case editor), this property setting enables you to test how various response maps will work. When you specify a particular response map (from the associated response map library), iTest applies the map to the response. You can then preview the blue boxes in the Response view and the results of queries in the Queries view. During test case execution, the value specified for **Response map name** is used and the **Response map to use at design time** setting is ignored.
Use an auto-generated response map if no other map is available	Check the box to cause iTest to apply auto-mappers to the response in the case that no applicable map is found. **Note** Even if you select the **Do not use a response map for this step** option, iTest will apply an auto-generated response map if you check the **Use an auto-generated response map if no other map is available** property.

Global analysis rules

Skip global analysis rules from session profile and test case	Use only the analysis rule that is defined for the step and do not use the global analysis rules that are defined in the test case or in the session profile associated with the step.

Store Response

For details on using the properties in this group, see "Storing a response into a variable (for use later in the test)" on page 140.

Step Properties section: Events properties group

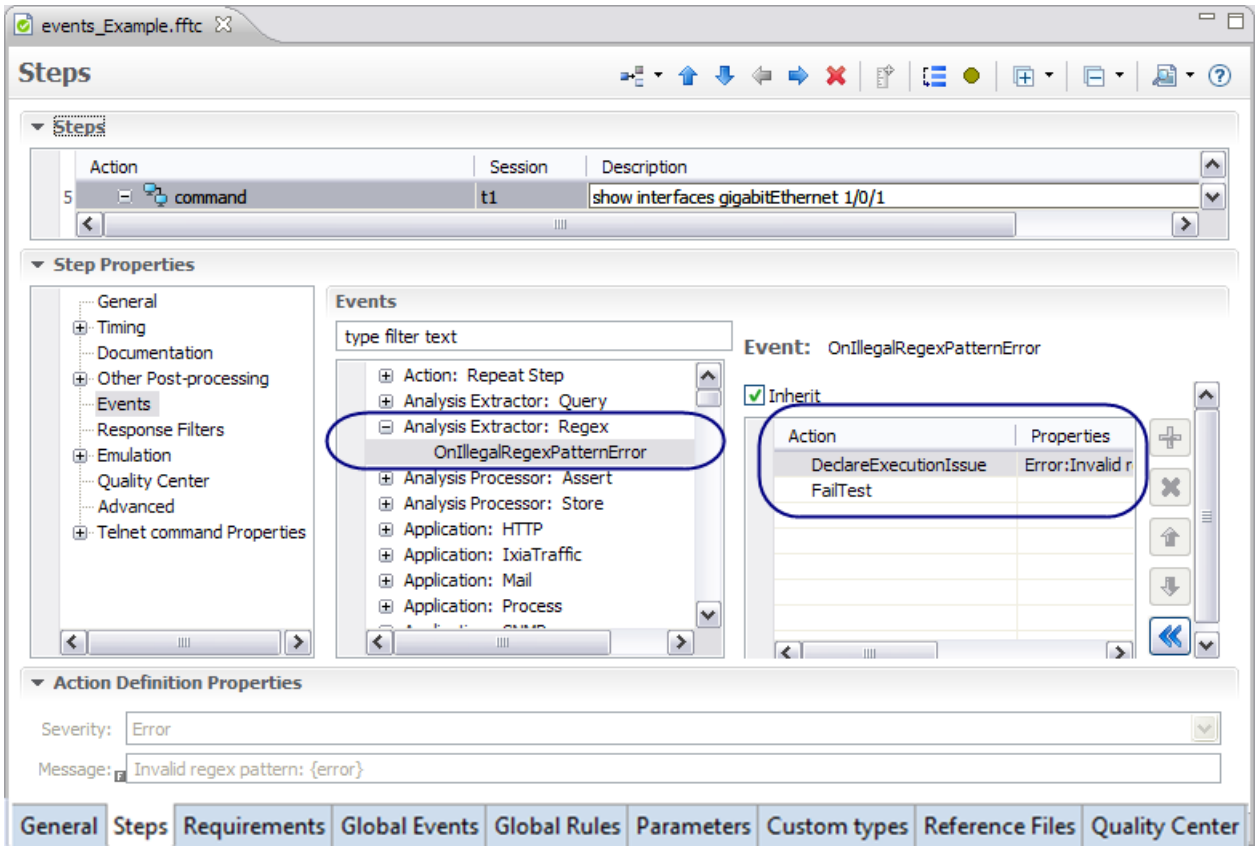
On this page, you specify the default actions that should take place whenever particular events occur during execution of the selected step. Events are described more fully in Chapter 27, "Events: Taking Action when a Particular Event Occurs During Execution".

Configuring the actions to take for events for a step

In this example, we have selected the **show interfaces** step. On the **Events** properties page for the step, in the **Analysis Extractor: Regex** group of events, we have selected the **OnIllegalRegexPatternError** event. When we select the event, the actions that are currently specified for the event are listed in the **Event** section's table of Actions.

The settings that you make on this page are similar in nature to the settings on the **Global Events** page — the difference is that the settings that you make here apply only to the current step. To

configure the actions to take for an event, follow the instructions in "Configuring Events: The Global Events page" on page 603.



Step Properties section: Response Filters properties group

You can use ***response filters*** to remove unwanted text from a response after a step has executed and before iTest applies analysis rules. The "filtered" response (the portion of the response that remains) is typically cleaner to read and easier to map and to understand. As a result, the sample responses that you use to create Response maps are simpler, and test reports are more readable.

For further information, see Chapter 25, "Filtering Unwanted Text from Responses".

Step Properties section: Emulation properties group

You can have iTest emulate ("pretend to be") any step or session. When you activate emulation for a step, then iTest does not send the command to the session as usual; instead, iTest just returns a response that you specify — the emulated response (typically, the response that the session returned the last time the test case executed — you can edit the response as needed). No interaction with a session or device occurs — iTest simply returns the 'canned' response.

For further information, see Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions".

Step Properties section: Quality Center properties group

iTest is strongly integrated with Quality Center. You can start test execution and monitor the results in both applications.

Note Quality Center features require a separate license and are supported on Microsoft Windows only.

For further information, see Chapter 47, "HP ALM (formerly HP Quality Center Server)".

Step Properties section: Advanced properties group

You can set the following advanced settings:

Default session	Use the **Default session type** property for steps that do not include a **Session** (typically set by an associated **open** step) — for example, in procedure libraries and other places where field replacement might not allow the step's session to be resolved. When you click **Browse**, the **Select Session Profile or Device** dialog box opens, as described in "Selecting a session profile or device" on page 1294.

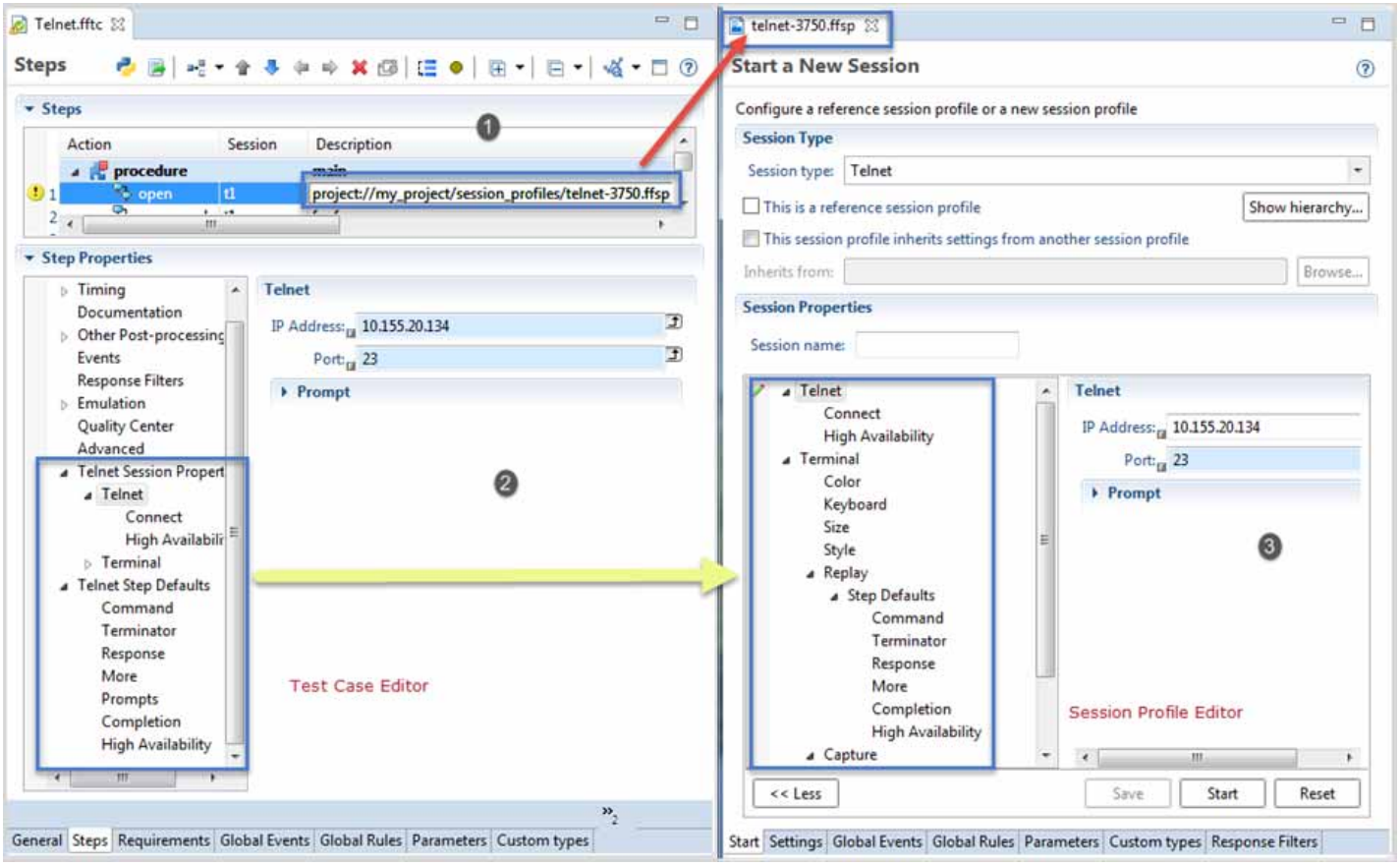
Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step

In test cases, **open** steps refer to property settings in testbed devices or session profiles to determine how to start a session. For an **open** step in a test case, you have the option to override any of the property settings so that all steps in the session use the new property settings. The set of properties that appears in the **<sessionType> Session Properties** section exactly reflects the set that appears in the testbed editor and the Session Profile editor (**Start** page).

We use a Telnet session in the example, but the properties for the **open** step exactly reproduce the properties in the testbed device or session profile for any session type.

Important The changes that you make to the property settings for the **open** step apply only during test case execution. The settings in the testbed device document or session profile document do not change.

1 The **open** step refers to the **telnet-3750.ffsp** session profile document. The set of **open** step properties is exactly the same as the set of properties that you define in the testbed device or session profile that is specified for the **open** step.



2 Any changes that you make *here*, in the **Telnet Session Properties** section of the Test Case editor . . .

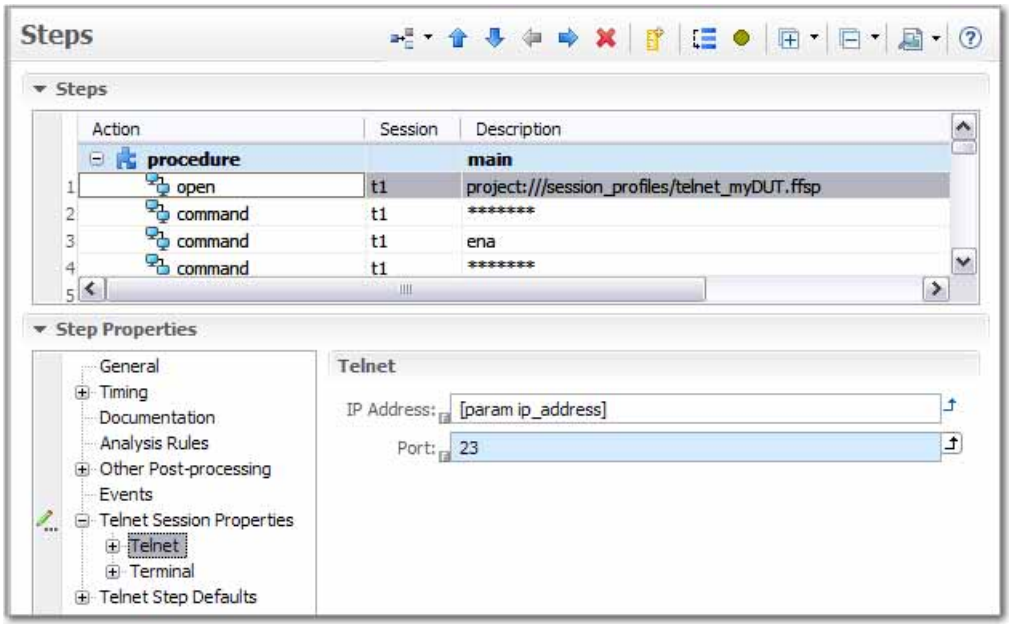
3 . . . overwrite the settings that you had previously made *here*, in the Session Profile editor.

4 The new settings apply to any step in session **t1** (and only for the duration of the session).

Using parameters in open steps

You can use test case parameters to overwrite property settings that you make in the session profile. As a result, the test case can make property settings dynamically set at runtime. For example, let's say you created a session profile named **telnet_myDUT.ffsp** with an **IP address** setting of **99.88.77.66**.

You then create a test case parameter named **ip_address** and set its value to **11.22.33.44**. In the Test Case editor, use the **param** command as a field replacement in the **open** step's **IP address** property setting:



As a result, at runtime, the **open** step's **IP address** setting overwrites the IP address specified in the session profile and the session opens using **11.22.33.44**.

Requirements page on the Test Case Editor

Test case editor: Requirements page

Define the Agent capability requirement to be used with the Test Case exported to Velocity. Use the **Requirements** page to add agent capabilitites requiremetns. (You can also delete and update the list.)

Important Requirements are associated with the test when exported as iTAR and imported in Velocity. The requirements will be available as options in the Agent requirements selection list.

Specifying Agent requirements

1. On the Test Case editor, open the **Requirements** page.
2. Click  to add a new attribute to the list.

3 Specify the following properties for an Agent:

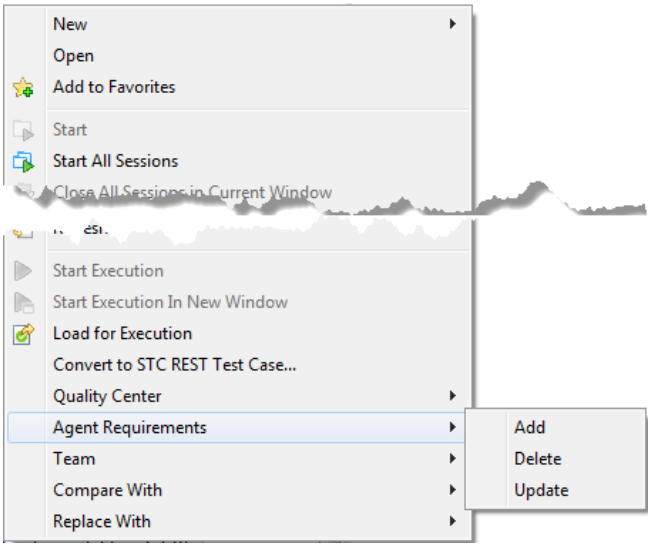
Key	The name that you use for the Agent must be unique. **Note** Use any name that allows you to identify the Agent capability. For example, os_type, agent_id, browser_type, etc. The following validation rules apply: • A Key name should not start with these special characters: **';', '#' or '['** • Value should not end with: **']'** • Both Key and Value should not contain: **'='** • Duplicate key and value pair is not supported, a warning displays saying that the value pair already exists.
Value	Specify the value of the associate Agent type, linux, debug_agent, chrome, etc.

Note The name/value pair you enter is not case sensitive. iTest converts the name and value pair to lower case.

Tools on the Requirements page

Add	Add a new key/value pair to the list.
Remove	Remove the selected key/value pair from the list.
Move Up / Move Down	Move the selected key/value pair one place up or down in the list. **Note** This feature is purely for your convenience in organizing the list and has no bearing on how the Agent Requirements are used.

You may also specify Agent Requirements to several projects with multiple test cases from the iTest Explorer right-click action menu. This action is particularly useful if you have a large workspace with several projects and test cases within each project. For example, you may select my_project, test_case folder, or multiple test cases with the folder, right-click, select the **Agent Requirements** option and add a common Agent Requirement, update an existing Requirement, or delete a previously applied Requirement.



Global Events page on the Test Case Editor

Test Case editor: Global Events page

The **Global Events** page appears in several editors. On this page, you specify the default actions that should take place whenever particular events occur during execution. Events are described more fully in "Events" on page 603.

See "Configuring Events: The Global Events page" on page 603

Global Rules page on the Test Case Editor

Test Case editor: Global Rules page

The **Global Rules** page appears in several editors. Use the page to define analysis rules that apply to all steps in a test case.

See "Global Analysis Rules page" on page 665

Parameters page on the Test Case Editor

Test Case editor: Parameters page

The **Parameters** page appears in several editors. Parameters that you define for a test case can be accessed by any step in the test case.

If you specify a Global parameter file, then the parameters that you define here are merged with the parameters in the Global parameter file.

See Chapter 33, "Parameters".

Custom Types page on the Test Case Editor

Test Case editor: Custome type page

In the Test Case editor, Testbed editor, or Session Profile editor, the **Custom type** tab allows you to define custom parameter types and their values.

See Chapter 33, "Parameters", section "Custom Types" on page 750

Reference Files page on the Test Case Editor

Test Case editor: Reference Files page

To fully define a test and to describe it for coworkers, you might want to associate files with the test (a config file, a topology diagram, and so on). Use the **Reference Files** page to add links to files that should be associated with the test case. (You can also delete and update the list.)

Important Reference files are associated with the test as supporting documentation and are not used in any way during execution.

Specifying reference files

1. On the Test Case editor, open the **Reference Files** page.
2. Click  to add a new file to the list.
3. Specify the following properties for the file:

Name	The name that you use for the reference file must be unique. **Note** Use any name that serves your purposes in communicating the relevance of the file — the name does not have to be the filename.
Location	Specify the URI of any file (inside or outside the workspace). Use **Browse** as needed. Click the **Location** link to open the file in an editor.
Description	Optional. Type one or more lines of text to describe the file for coworkers.

Tools on the Reference Files page

Add	Add a new file to the list.
 **Remove**	Remove the selected file from the list.
 **Move Up / Move Down**	Move the selected filename one place up or down in the list. **Note** This feature is purely for your convenience in organizing the list and has no bearing on how the files are used.

4. **Quality Center users:** If you publish the test case to Quality Center, then all reference files are listed on the **Test Instance Properties** page in the **Test Lab** for the test. The names appear as **attachmentName_fileName**

Quality Center page on the Test Case Editor

Test Case editor: Quality Center page

iTest is strongly integrated with Quality Center. You can start test execution and monitor the results in both applications.

Note Quality Center features require a separate license and are supported on Microsoft Windows only.

For further information, see "Test Case editor: Quality Center page" on page 957.

Setting preferences for the Test Case Editor

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Editors > Test Case Editor.**

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Properties in: Spirent > Editors > Test Case Editor

Ask for confirmation when starting capture to test case	Start Capture starts direct-to-test case capture. Check the box to request confirmation to start the direct-to-test process. Default: checked
Display a warning when the user renames the main (entry point) procedure	If you rename the entry point to a test case (typically, the entry point procedure is named **main**), the test case will not start. This is not an issue if the file is a library of procedures that are meant to be called from test cases. (See "Creating a procedure 'call' step using in-line editing" on page 231) Check the box to warn users when they rename the entry point. Experienced users that are creating procedure libraries will uncheck the box. Default: checked
Display Step IDs	Check the box to include a column titled **Step IDs** in reports. Default: checked
Perform step validation only when requested	By default, iTest auto-validates steps as you create them. Validation determines whether there is a a problem with a step and whether any property settings are non-default. Check the box to cause iTest to validate test case steps only when you click **Validate** in the toolbar. Default: unchecked

Properties in: Editors > Test Case Editor > Appearance > Colors

On this page, you specify the various colors displayed in the Test Case editor.

Properties in: Editors > Test Case Editor > Errors/Warnings

On this page, you indicate that the Test Case Editor displayes a warning when the mandatory values and default values have not been specified.

Severity levels displayed in the Problems view:	Indicates the message displayed in the Problem view is as per the level selected. For example, If you select Error, Warning, or Info, a message displays saying: Optional/Required argument(s) with default value(s) are not specified.
Default values(s) will be used for optional arguments(s)	Set the severity level to indicate that a procedure or QuickCall will use default values for the *optional* argument(s). That is, the argument value was not supplied, but a default value exists for the procedure or QuickCall. Options: Ignore, Error, Warning, Info Default: Ignore
Default values(s) will be used for required argument(s)	Set the severity level to indicate that a procedure or QuickCall will use default values for the *required* argument(s). That is, the argument value was not supplied, but a default value exists for the procedure or QuickCall. Options: Ignore, Error, Warning, Info Default: Ignore

The severity level indicated are processed as follows.

Process	**If empty required argument, no default value...**	**If empty required arg, default value exists...**	**If empty optional argument, default exists...**
validation	Displays validation error	As per the setting in Wndows>Preferences for the required argument	As per the setting in Wndows>Preferences for optional arguments
execution	Displays warning message	Displays warning message	Displays warning message

Properties in: Spirent > Editors > Test Case Editor > EXEC Steps

Automatically add 'then' and 'else' clauses on new 'if' steps	Auto-populate the full structure of new **if** steps with hints that help you to create a working loop. Default: checked
Automatically add nested placeholder steps inside container steps (if, while, for, foreach)	When you add a flow control step, iTest can auto-populate steps that act as placeholders for the steps you will eventually add. Notice that the steps are indented under the container step to indicate that they are a part of the loop construct. Skipping, moving, or otherwise editing the container step affects all steps in the loop. Default: checked
Display warnings in the Step Issues view for analysis rules that cannot be evaluated at design time	When an analysis rule includes a field replacement whose value can be determined only after particular steps in the test case execute, then, while you are editing the test case, iTest cannot guarantee that the field replacement will operate successfully. Check the box to generate warnings in these cases. Default: checked
Display a warning when the user adds a got step or goto action is added	This option applies to imported Spirent SVT steps only. When you add a **goto** step or action, iTest displays a warning to notify you that the **goto** action is deprecated (no longer an active part of iTest). The warning appears both as icon in the left column of the Test Case editor's **Steps** page and an entry in the Step Issues view. The warning reminds you that adding **goto** steps is poor practice for software development. Uncheck the box to disable such warnings. Default: checked
Display the full URIs for foreign procedure calls	Default: unchecked While editing a **call** step or **CallProcedure** step that calls a foreign procedure, iTest displays the URI of the test case (procedure library) that contains the procedure definition. This helps you to identify the correct procedure library so that you can select the correct procedure and then finish defining the calling step. If this option is unchecked (default), then, after you finish editing the command in the Description cell (adding the procedure name and any arguments) and exit the cell, the full URI to the procedure library will no longer appear — only the procedure name plus arguments remain. This makes it easier to read and understand the step. If you check the box, then the full URI to the procedure library (test case) remains on display in the **Description** cell.
Display a warning when inserting call steps with missing required arguments	While editing the arguments for a **call** step or **CallProcedure** step (either in the **Description** cell or while using the Procedure Call wizard), it is possible to define a call that does not include all required arguments. Check the box to display a warning in this case. Default: checked

JSON Editor

Overview

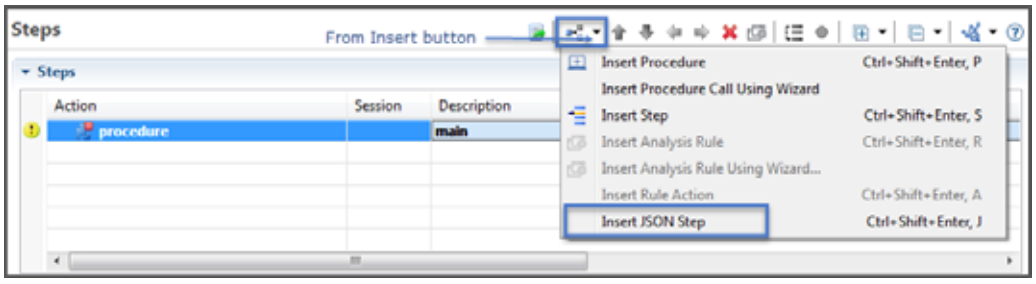
iTest provides JSON Editor to easily formulate a JSON document and also manipulate the JSON response. That is:

- Formulate a JSON document from the TestCase Editor (See also Chapter 8, “Test Case Editor”). A Wizard provides an easy point-and-click way to perform CRUD operations on nodes in a JSON document.
- Manipulate the JSON response. iTest provides a programmatic way of performing CRUD operations on **block response** of a procedure (see also Chapter 10, “Procedures”, section “Defining a procedure” on page 223).

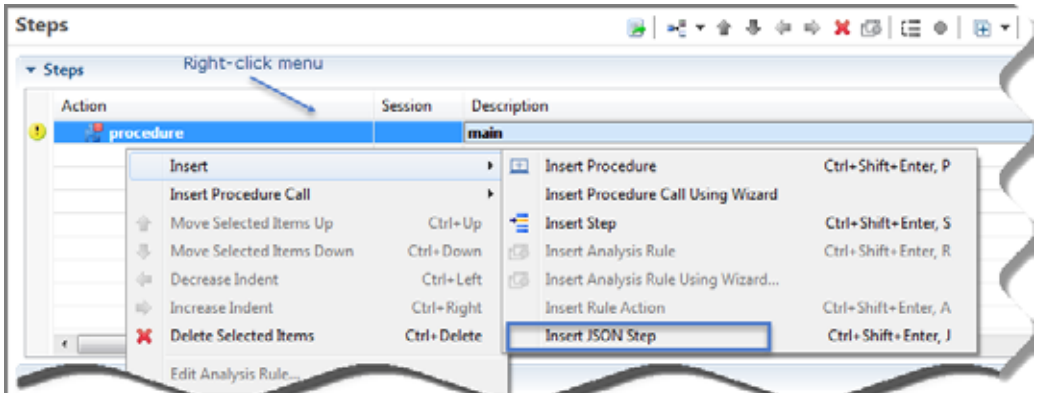
Insert JSON Step

While adding a step in the Test Case editor, click in the **Action** cell to open the **Insert JSON Step wizard** by using one of the following methods:

- Click the **Insert** button on the **Test Case Editor** and select **Insert JSON Step** to open.



- Right-click context menu sub-menu items



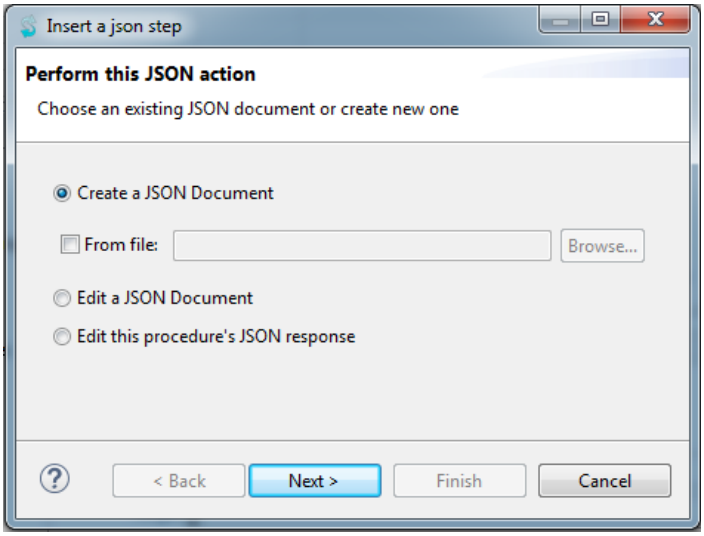
- Use the quick access key Ctrl + Shift + Enter, J

The **Insert JSON Step** wizard opens.

JSON Editor Wizard

The JSON editor allows you to select a JSON document to edit perform these actions.

Note A JSON document is a variable in iTest. The step above (createJson) is just a step that creates this variable from some starting point.



- "Create a JSON document" on page 195
- "Edit a JSON document" on page 198
- "Edit this procedure's JSON response" on page 199

Important Once you have competed including all the information as required on the wizard, only one JSON command will be inserted into the test case. See "JSON Command in Test Case" on page 200.

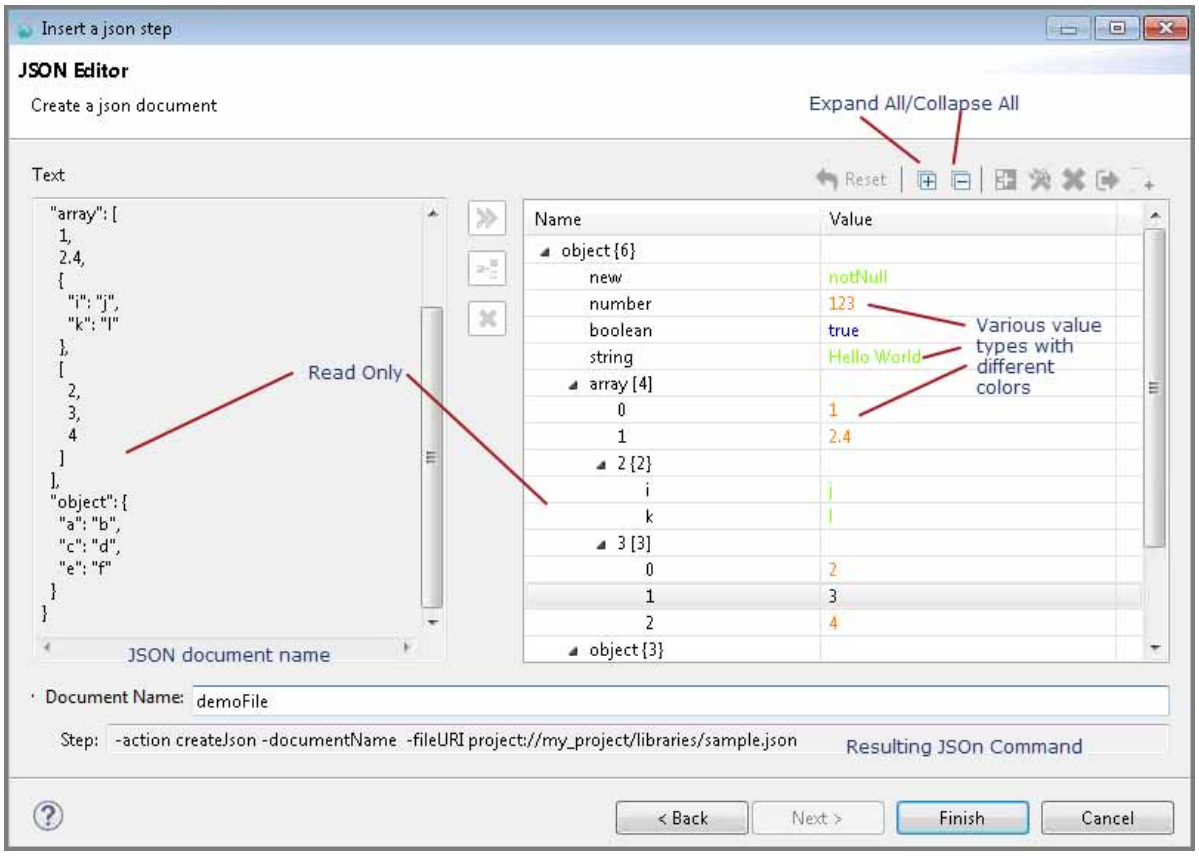
Create a JSON document

Select the option **Create a JSON document** and click **Next** to create a new JSON document from an existing JSON document or create one manually from JSON string.

From file: Select to create a new JSON document from an existing project or file URI.

When creating a JSON document from file, if you choose a document that contains one or more JSON syntax errors, an error displays and you will not be able to click **Next** to view the document. The Next button is available only when a valid JSON file is chosen (when no syntax errors exist).

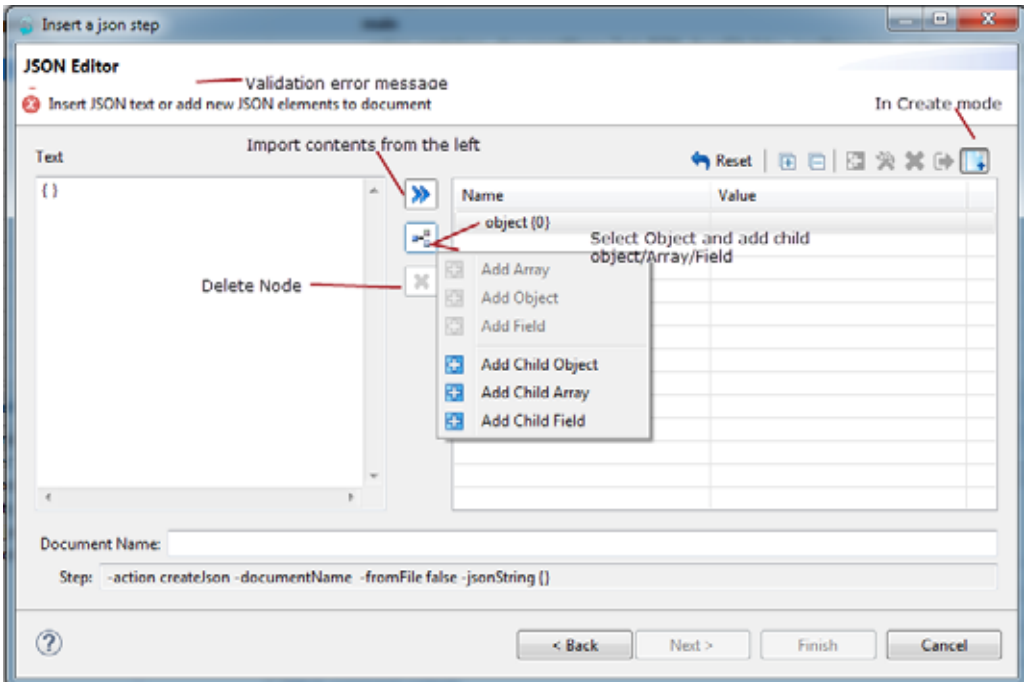
- **Selected:** When selected, the JSON editor displays the contents of the file (both in pretty print and expand/collapse structure).The file contents will be displayed in the JSON editor but no edits to the document will be allowed.



You will then be able to “name” your JSON document and the wizard will finish, adding one step to the test case, something like:

```
json -action -documentName myJsonDoc -fileURI -contents
project://test/sample.json
```

- **Not selected:** (manually create JSON document from JSON string) The JSON editor will display empty text and structure, allowing you to paste text into the editor and make changes as required.



Note This options allows you to Create a JSON document, i.e., add, update, and delete json node.

You will then be able to “name” your JSON document and the wizard will finish, adds one step to the test case. For example:

```
json -action -documentName myJsonDoc -fromFile false -jsonString
{“name1”:”value1”}).
```

Note If you insert custom (your required) JSON text on the text area and click the **Import** button, the JSON command will be generated with the structure of this text.

Note The resulting json command will appear in a preview area of the wizard. Click **Reset** to clear any change you made and put the wizard back to the when the editor opened (allowing you to perform a different operation on the JSON document).

Important Once you have competed including all the information as required on the wizard, only one JSON command will be inserted into the test case. See “JSON Command in Test Case” on page 200.

JSON Editor Buttons

Action Buttons	Description
Reset	Resets JSON content to its original content
Import	Imports content from left panel to right panel.
Expand All/Collapse All	Expands/collapses all nodes of the current JSON structure tree.
Insert:	Inserts various types of JSON nodes.
Array	Inserts a new row with empty key into current selection in JSON Tree. This row will contain sub-item which have the same key “item” and value will be empty. **Note** The key in array cannot be modified.
Object	Inserts a new row with an empty key, in the current selection in the JSON Tree. This row contains no key and value, if there are child objects.
Field	Inserts a node with empty key and value.
Child Object	Inserts a new empty object nested within the current selected Object/Array node in the JSON tree.
Child Array:	Inserts a new empty array nested within the current selected Object/Array node in the JSON tree
Child Field	Insert a new field with empty value nested within the current selected selected Object/Array node in the JSON tree

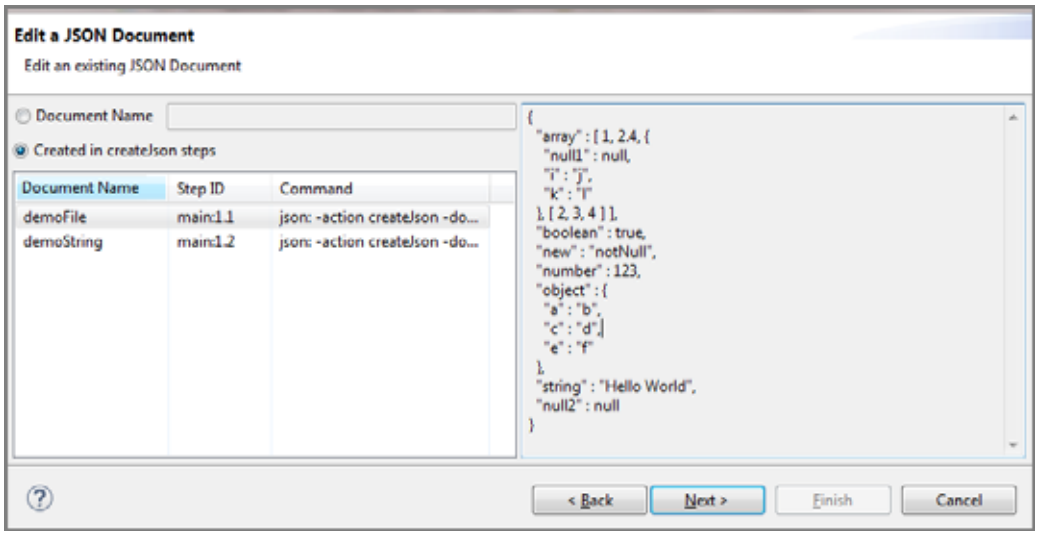
Note The different value types have different color. The color of each value type will be configured by default as follows.

- Number: Red
- String: Green
- Boolean: Blue

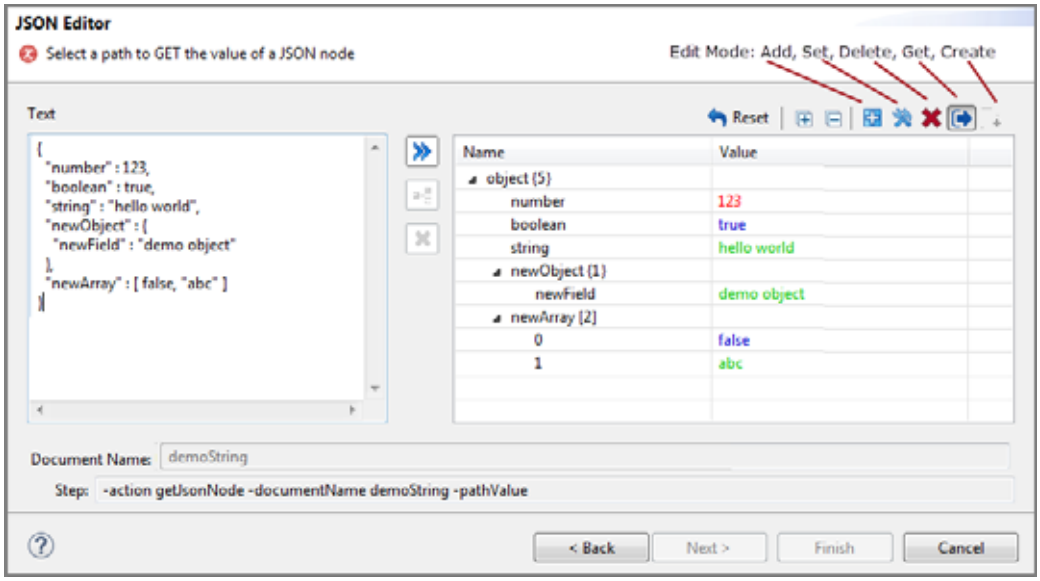
You may configure the color on the “Setting preferences for JSON Pretty Print” on page 203.

Edit a JSON document

The Edit Allows you to edit an existing JSON document. Select option and click **Next**. The wizard displays a list of JSON document(s) created in the step above (**createJson**), and allows you to reference a custom variable via the **Variable** text box.



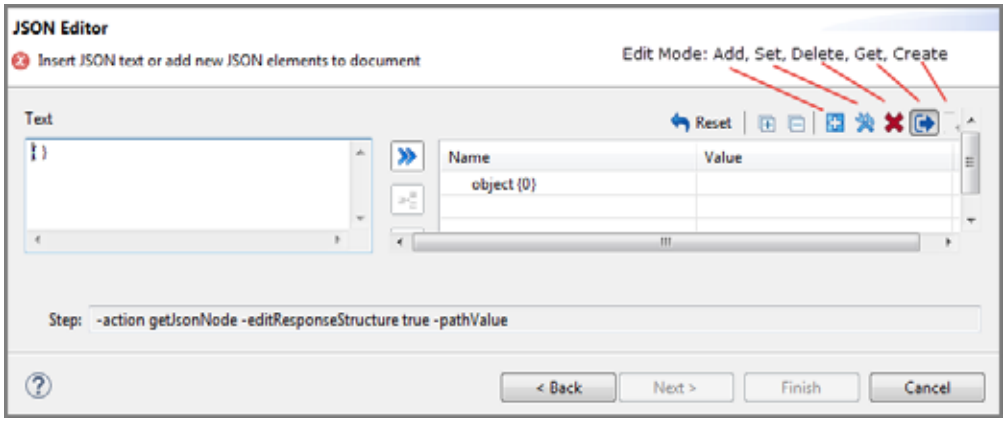
- If you select a JSON document created using **createJson**, then the wizard will display its initial contents (the document as it existed at creation time) in the editor (both in pretty print and expand/collapse structure). See "JSON Editor Buttons" on page 197.



When path is empty on Edit, the message displayed depends on the mode selected. For example:

- Select a location in the document and SET a new value
- Select a location in the document to GET the value of a JSON node
- Select a location in the document and click ADD
- Select a location in the document and click DELETE

- If you enter a custom variable, the wizard will display empty text and structure. You may paste raw text on the left pane, the editor will render the JSON structure on the right pane, and then allow one operation (read/write) on the JSON document. See "JSON Editor Buttons" on page 197.



Irrespective of whether you entered a custom variable or selected a document created by **createJson**, the JOSN Editor Wizard allows you to insert alternate JSON in the text pane and perform one of these operations on the document: **getJsonNode**, **setJsonValue**, **deleteJsonNode**, or **addJsonNode**.

Note If you insert custom (your required) JSON text on the text area, the JSON command will be generated with the structure of this text.

Note The resulting json command will appear in a preview area of the wizard. Click **Reset** to clear any change you made and put the wizard back to the when the editor opened (allowing you to perform a different operation on the JSON document).

Important Once you have competed including all the information as required on the wizard, only one JSON command will be inserted into the test case. See "JSON Command in Test Case" on page 200.

Edit this procedure's JSON response

The **Edit this procedure's JSON response** option allows you to edit the procedure's JSON response (defined in "Defining a procedure" on page 223 in Chapter 10, "Procedures"). An error displays if the current procedure does not define JSON Response.

Note You are not prompted to select a JSON document or enter a variable value as you wish to read/write the procedure's response structure.

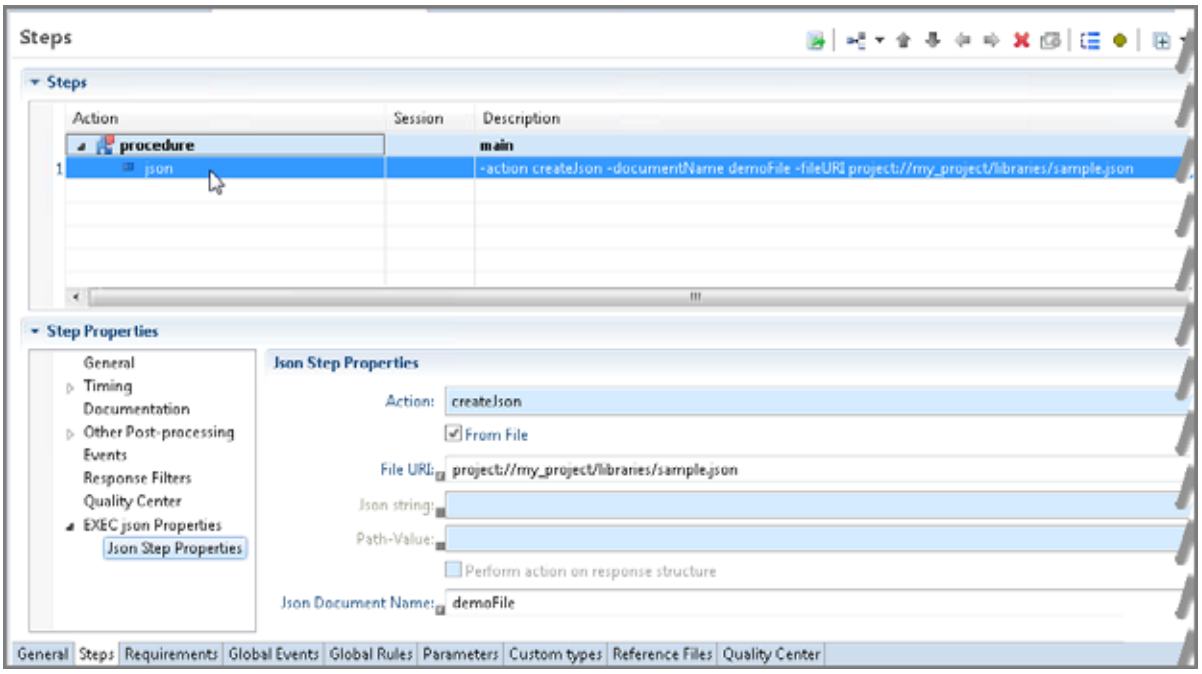
When the editor opens, it will display the initial contents of the response structure (the contents statically defined in the procedure definition). You may paste (overwrite) raw JSON text on the left pane and perform one operation on the JSON document. See "Edit a JSON document" on page 198.

Note The resulting JSON command will appear in a preview area of the wizard. Click **Reset** to clear any change you made and put the wizard back to the when the editor opened (allowing you to perform a different operation on the JSON document).

Important Once you have competed including all the information as required on the wizard, only one JSON command will be inserted into the test case. See "JSON Command in Test Case" on page 200.

JSON Command in Test Case

Once you have competed including all the information as required on the wizard, only one JSON command will be inserted into the test case.



iTest supports five action types for the JSON step.

JSON Action	Example
createJson	Creates new JSON document from file URI or JSON string. `json -action createJson -documentName myJsonDocument -fileURI "project://my_project/jsonFiles/json.txt"` Creates from JSON String: json -action createJson -documentName myJsonDocument `-jsonString {"name1":"value1"}`
setJson	Defines parameter with "original json" string and "newValue". `json -action setJsonValue -documentName myJsonDocument -pathValue {"true":true, "false":false, "nullValue": null}` **Note** To modify a JSON key, you may delete an existing key and insert a new key.

JSON Action	Example
getJsonNode	Get the value of a JSON node `json -action getJsonNode -documentName myJsonDocument -pathValue {“object/c/[1]”}` **Note** pathValue is a string of the path. Gets value of only one JSONnode.
deleteJsonNode	Deletes an existing JSON `json -action deleteJsonNode -documentName myJsonDocument -pathValue {“key1/[0]”, “array/[4]/[1]”, “myKey1”, “myKey2”}` **Note** pathValue is an array of key paths (no need JSON object syntax is required).
addJsonNode	Add new node into a JSON document `json -action addJsonNode -documentName myJsonDocument -pathValue {“/”:{“newNode1”:”newValue1”, “newNode2”:{“n2”:”V2”}}}`

Note The arguments would appear both in the **Description** column of the test case and the **Step Properties**, and be synchronized.

Json command does not support command field level substitution.When using filed replacements, ensure that you enable field substitution from the sections:

Step Properties > EXEC json Properties > Json Step Properties.

See also Chapter 28, “Field Replacements” and Chapter 8, “Test Case Editor”, section “Step Properties section: General properties group” on page 176.

Manually setting JSON command line arguments

When you manually insert a JSON step (when editing a Test Case) a popup of arguments with their descriptions displays under these circumstances:

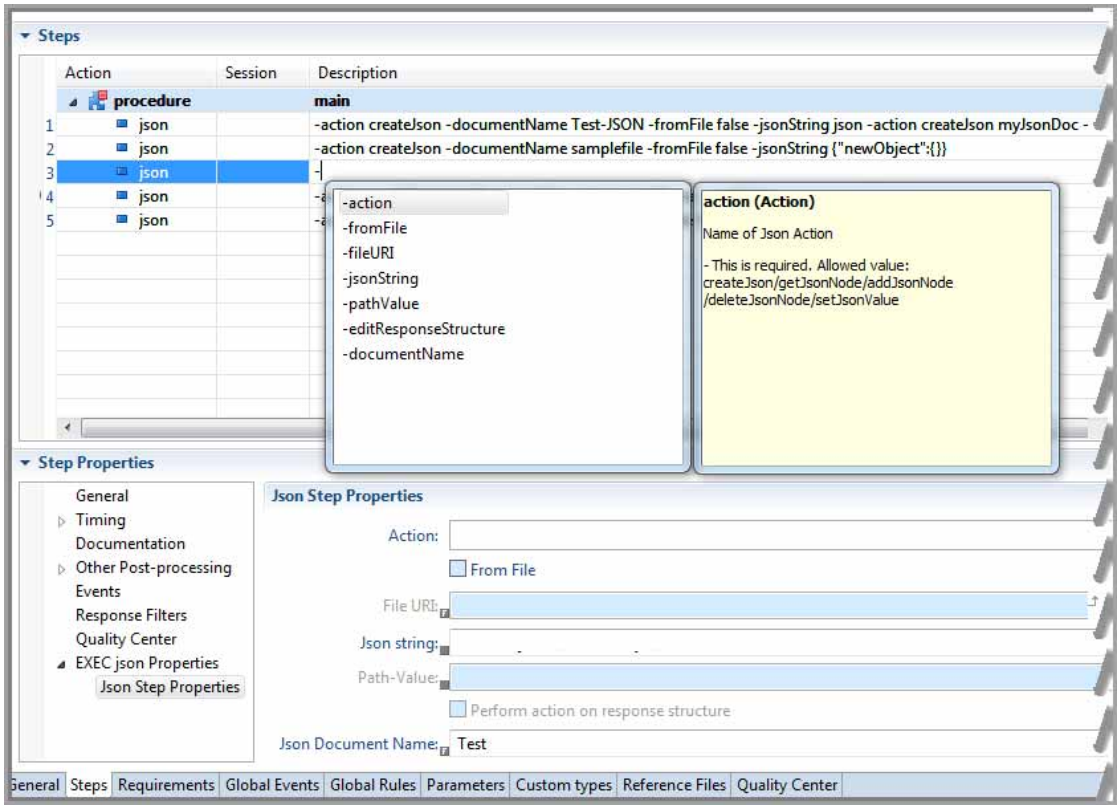
- When you enter a JSON action as a step and the description field is empty, press **Ctrl + space.**
- When you enter a JSON action as a step, then enter a hyphen (**-**) in the description.

Note No validation is performed for the pathValue syntax when you manually enter a pathValue argument. Your input value will be automatically be filled under the Step Properties. The pathValue under Step Properties is validated for correct syntax. If you input invalid syntax, this field will be flagged with the error.

The following rules apply when parsing the manually enetered description:

- If the set of arguments is invalid, the Test Case Editor erases the text typed into the description field and the Step Properties window will not be updated.
- If the set of arguments is valid, the Test Case Editor updates (synchronizes) the Step Properties window.

Note The JSON step executor only uses the Step Properties for execution. Description field and step command field is only used for parsing value to Step Properties.



iTest support these command line actions to be entered manually for the JSON step.

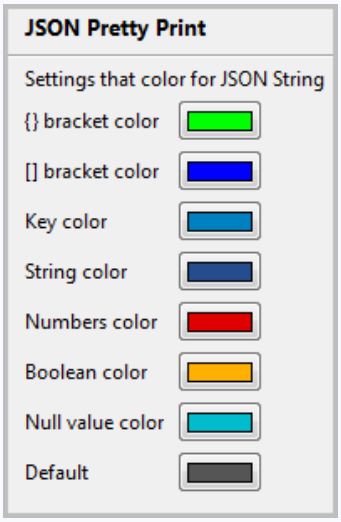
Command	Description
-action	Enter the name of Json Action Mandatory. Allowed values: createJson, getJsonNode, addJsonNode, deleteJsonNode, setJsonValue
-fromFile	Create json document from an existing file. Allowed value: true/false This is enabled only when action = createJson
-fromURI	File path used to retrieve file. This is enabled and required if action = createJson and fromFile = true.
-jsonString	Original Json content which is input manually. This is enabled and required if action = createJson and fromFile = false.
-pathValue	Path-Value used to query json node. This is enabled and required if action = getJsonNode, addJsonNode, deleteJsonNode, setJsonValue.
-editResponseStructure	Perform action on response structure. Allowed value: true/false This is enabled if action = getJsonNode, addJsonNode, deleteJsonNode, setJsonValue.
-documentName	Json Document Name. Mandatory.

Setting preferences for JSON Pretty Print

To view or edit settings that JSON editor display, click **Window > Preferences.** On the **Preferences** page, click **Spirent > JSON Pretty Print.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > JSON Pretty Print

Setting colors for JSON String	Different value types may be set to have different color. The color of each value type will be configured by default as follows. When JSON format is selected or auto-detected, iTest will apply pretty print formatting and assign different colors as selected to: • Strings • Numbers • Special values: true, false, null, etc.
	Click the color shown to display a color palette and select a required color.  iTest will use different colors for characters like { },][, =, etc. iTestRT and Velocity agent also auto-detects JSON response data, and if the response is valid JSON, iTest formats the response as JSON prettiy print.
Apply	Click Appy to save the changes or Cancel or Restore Defaults to discard the changes.

CHAPTER 9

QuickCalls: Defining and using a library of custom actions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Overview: QuickCalls

Each iTest session type supports built-in actions. For example, Telnet and SSH session profiles support the **break** and **command** actions. SNMP session profiles support actions like **get**, **getNext,** and **walk**.

iTest allows you to add custom actions to those built-in actions. *QuickCalls* are custom actions that ***you*** define for a session profile or device. QuickCalls make it easy to execute a predefined set of steps — either as a single QuickCall step in a test case or as a single button click in an interactive session.

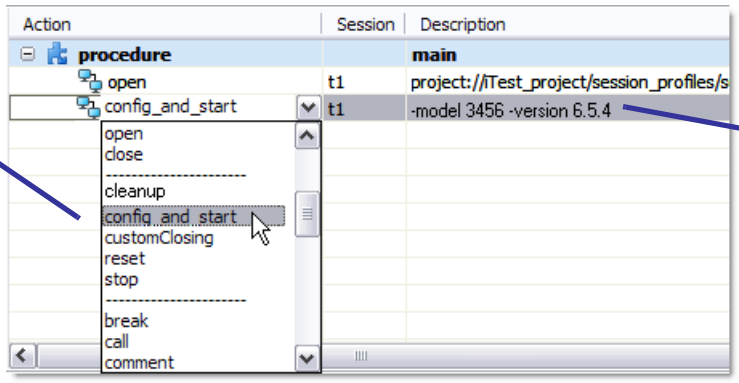
Important A test case defined in one language can call procedures and quickcalls from files defined in other languages. For example, a test case written in Python syntax can issue quickcall commands from files written in TCL syntax.

QuickCalls in test cases

In the Test Case editor, QuickCall names appear in the drop-down list of **Actions** for any step in the session (just like the built-in actions for the session). The QuickCalls appear as a separate group of actions after the **open**/**close** actions and before the EXEC actions. To add a QuickCall step to a test case, select the QuickCall from the list.

These are the QuickCalls.

(As you will read in a moment, they are defined in the QuickCall library referenced by the session profile that started session **t1**.)



iTest suggests argument values for the QuickCall. This information comes from the QuickCall definition that you or a coworker configured.

QuickCalls in interactive (manual) sessions

During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step.

How iTest captures and reports on QuickCalls

iTest captures every QuickCall that executes. When you save a captured session as a test case, each QuickCall becomes a single step in the test case. This improves readability, portability, and consistency. In test reports, you can expand a QuickCall step to view the individual steps that executed to perform the QuickCall (identical behavior as with procedures).

Defining QuickCalls

You define QuickCalls the same way that you define procedures using the Test Case editor. You add a procedure definition, add all of the steps into it (you can save the steps from captured interactive/manual sessions), and then (typically) set the procedure as “public” (that is, include the QuickCall name whenever a user asks to see a list of available QuickCalls). Repeat for as many QuickCalls as needed. When you save the test case that holds the QuickCall definitions, you are saving a *QuickCall library*. QuickCall libraries are “public” — the editor for any testbed device or session profile enables you to specify that the QuickCall library is associated with the session configuration.

QuickCalls are associated with session profiles. To make the QuickCalls in the library available in a session, you specify the library on the **Libraries** properties page in the Testbed editor or on the **Misc** page of the Session Profile editor. As a result, the QuickCall is available:

- **During interactive sessions**. Click QuickCall and select the action from the drop-down list. All of the steps in the QuickCall execute. (If the QuickCall requires parameter values, then iTest opens a dialog box to allow you to supply the values.)
- **While you are editing the test case**, as shown earlier
- **During automated execution**. The QuickCall executes exactly like a procedure executes.

Limitation Global variables defined in one QuickCall in the library cannot be accessed by other QuickCalls in the library.

How QuickCalls execute

QuickCalls are procedures and they execute like procedures. iTest captures each QuickCall execution as a single high-level action (just like a keyword). So, when you save a captured manual session as a test case, each QuickCall is added as a single step. The resulting test case is much more readable and understandable than it would have been if each step that makes up the QuickCall had been listed.

iTest does not enforce timeout settings for any **call** step associated with a QuickCall.

In test reports, you can expand a QuickCall to view all of the steps that executed.

For other details, see “How procedures execute” on page 220.

Note If a **call** step in a child test case **B** (begun by a **run** step in a grandparent test case **A**) calls grandchild test case **C**: The **call**ed test case **C** will use the shared session from test case **A** in its **open** step if the **Session ID** in **C** is same as the **Session ID** in **A**. If you do not want to use the shared session, then change the **Session ID** in **C** to be different from the **Session ID** in **A**.

Parameters in QuickCalls

Parameters in QuickCalls are resolved as follows: Say that session A inherits from session B and both A's QuickCall library and B’s QuickCall library have parameters. If we use any QuickCall from session A (even if the QuickCall is defined in session B and given to session A by inheritance), then the parameters are resolved “down the inheritance chain” — iTest first merges the parameters from A's QuickCall Library, then merges the parameters from B's library.

Note For procedures, parameters are resolved differently. See “Parameters” on page 222

Defining QuickCalls

Guidelines for creating QuickCalls

You create a QuickCall exactly the same way as you create a procedure.

- For ease of use and to improve portability and maintainability, define all related QuickCalls in a QuickCall library (described in a moment).
- Each QuickCall must have a unique name.

  Important Do not name a QuickCall “**procedure**” or “**Procedure**”.

- QuickCalls can call QuickCalls and/or procedures in a nested fashion.
- You can copy and paste QuickCalls from one library to another.

Defining a QuickCall

Step 1 First, specify the QuickCall library to associate with a device or session profile

1. You start the process from the testbed device or session profile that you want the QuickCall library to be associated with:
   - In the **Device Properties** section on the Testbed editor, open the **Libraries** page.

     or

   - On the Session Profile editor, open the **Misc** page.
2. In the **QuickCall** section, click the **Create QuickCall library** link.
3. The **New QuickCall Library** wizard starts. Specify a location and filename for the new library.

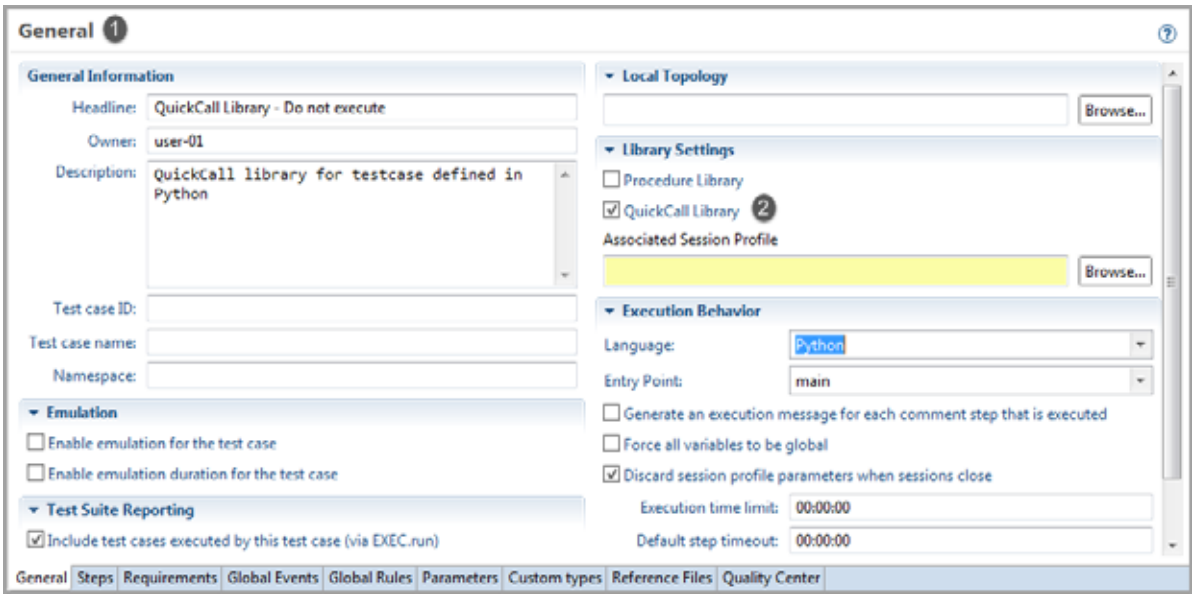
Tip Best practice is to place the QuickCall library in the same folder as the session profile that it is associated with and with the same filename. For example, **router3A.ffsp** and **router3A.fftc**

4 Click **OK**. The new QuickCall library opens in the Test Case editor

Note Once you have specified a QuickCall library, the link changes to **View QuickCall library**. Click the link to open the QuickCall library in the Test Case editor.

Step 2 Next, set up the QuickCall library

On the **General** page of the Test Case editor:



1 ***Document the library:***

In the **General Information** section, specify:

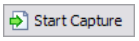
Headline	Optional. Type a one-line description that documents the usage and function of the QuickCall library. This text appears in the Favorites view to help you when selecting a QuickCall library. **Tip** You might include text that warns developers that this test case is not intended for execution.
Owner	Optional. Type the name of the person responsible for developing and/or maintaining the QuickCall library.
Description	Optional. Type additional text that describes the test case to make its usage clear to coworkers. **Tip** You might include text that warns developers that this test case is not intended for execution.

Make the QuickCalls "public" and associate them with a testbed device or session profile:

Include this test case when listing QuickCall libraries	Check the **Include this test case** check box to make the library "public", that is, to cause iTest to display the test case name whenever a user asks to see a list of available QuickCall libraries. This settings adds the QuickCall library to the drop-down list when you edit a session profile or device to associate with the library. Default: checked for QuickCall libraries and procedure libraries, unchecked otherwise
Session profile or device	Specify the testbed device or session profile to associate with the QuickCalls that are defined in the library. Once you have specified a device or profile, the link becomes active and opens the item in the appropriate editor.

Step 3 Now, define the QuickCalls that make up the library

Use the following instructions to add as many QuickCalls as needed to the QuickCall library. You define a QuickCall in either of the following ways:

- The easiest way to add a QuickCall is to manually execute the steps that you want to include in the QuickCall and then, using the capture-to-test feature 

or from the Capture view, save the captured session as a procedure in a test case. The QuickCall is added as a procedure definition after the last step in the test case. This option is described in "Adding captured steps into a test case or Python Script" on page 100.
- Alternatively, while working in the Test Case editor, you can add a QuickCall "manually" by adding a procedure definition and then adding steps to the procedure.

Tip While working on QuickCall definitions on the Test Case editor **Steps** page, click **Collapse All** to view only the QuickCall names and not the individual steps. You can then work on a single QuickCall definition without the clutter.

CAUTION Do ***not*** use the **open** action (open a session) in a QuickCall.

1 In the Test Case editor:

- If you added the QuickCall by adding a captured session, then select the procedure step (the step with an **Action** of "**Procedure**" that you just added). (Remember that each QuickCall definition is a procedure definition.)
- If you will add the QuickCall manually, select a step, click **Insert**  , and then select **Insert Procedure**. iTest adds a blank procedure definition below the selected step. (Remember that each QuickCall definition is a procedure definition.)

2 The predefined variable **$session** refers to the session in which the QuickCall is being invoked. In the **Session** cell, specify a session ID of **$session**. Because you specify a session ID of **$session**, and not a hard-coded session ID, you can call the QuickCall from any session. (Any session that calls the QuickCall passes its session ID to the quickCall steps using the **session** variable.)

Note In QuickCall libraries, **$session** is a reserved word.

3 In the **Procedure Properties** tree, click **General.** On the **General** page, specify values for the following properties.

Note

While working on a test case in the Test Case editor, right-click and on the menu displayed, select (**Show Properties View**). The Properties view appears as a tabbed pane on the iTest window.

The **Properties View** pane display is context-specific and varies depending on where your cursor is positioned. You may edit step properties using either the **Step Properties** section (within the Test Case Editor) or via the **Properties View** tab.

For example, if your cursor is at the **procedure** label (**Steps > Action> procedure**) and then select from the right-click menu, the **Properties** pane opens to display **Procedure Properties** (which is also displayed below the **Steps > Actions** section).

Note You may position and resize the tabbed Properties view window as required.

Note The text that you provide for the **Name**, **Headline**, **Author**, and **Version** properties (and for the **Description** property specified on the **Description** properties page) are displayed on the popup help for the QuickCall. The popup help appears when the test developer starts to type a QuickCall in the **Action** cell or types Ctrl+Space while in the **Description** cell for a QuickCall step.

Name	Required. Provide a meaningful name for the QuickCall. This string appears: • In the Test Case editor In the drop-down list of QuickCalls in the **Action** cell • During an interactive session, in the drop-down list of QuickCalls that appears when the user clicks QuickCall • In the Favorites view, under the associated QuickCall library **Note** Do not use the name of an existing iTest action (for example, **comment** or **readFile**). If you do, then during execution, iTest generates an Execution Issue Warning and then executes the built-in action.
Include this procedure when listing callable procedures	Check the box to display the QuickCall's name in: • The drop-down list of QuickCalls in the **Action** cell that appears when a test developer is adding a QuickCall step to a test case. • The drop-down list of QuickCalls that appears when the user clicks QuickCall during a manual session • The Favorites view (indented under the QuickCall library) This option makes the QuickCall available for use in any test case in the current workspace. Software developers call this “making a procedure public”. Default: checked for QuickCall libraries and procedure libraries, unchecked otherwise
Headline	Optional. Type a single line of text that describes the QuickCall. This string will appear with the QuickCall name in the drop-down list of procedures in the **Action** cell for a step . This text also appears in the **Headline** column of the Favorites view to help you when selecting a QuickCall.

Author	Optional. Type the name of the person who created the QuickCall definition. This helps coworkers who have questions about the QuickCall.
Version	Optional. If you track changes to the QuickCall definition, use this property to specify the version number of the QuickCall.
Response Map	Optional. Specify the response map to apply to the value returned by the QuickCall (via a **return** step in the QuickCall).
Default session	Optional. Because the **Session ID** of the steps in a QuickCall definition are typically parameterized so that the **Session ID** value can be set at runtime, you specify the testbed device or session profile URI here. This enables iTest (at design-time, while you are designing the QuickCall) to populate the **Action** cell with actions that are appropriate for the session type. If you are working in a QuickCall library and the **Session ID** is specified as **$session**, then this property defaults to the URI that is specified in the **Session profile or device** property on the **General** page of the Test Case editor.

4 In the **Procedure Properties** tree, click **Description**. For the **Description** property, type the text that should appear in the help popup for the QuickCall. We recommend that you show all forms of the QuickCall (for example, provide examples of all permitted combinations of required and optional arguments). Show each form on a new line.

5 QuickCalls can use both named arguments and numbered arguments. Perform this step if the QuickCall uses named arguments. The order of arguments in the list that you create here is the order in which they appear when a test case developer is adding a step that performs a QuickCall action.

For more detail on how QuickCalls use arguments, see “About arguments in QuickCall steps” on page 216.

In the **Procedure Properties** tree, click **Arguments**. On the **Arguments** page, click to add a new argument definition.

Specify values for the following properties for arguments:

Name	Provide a meaningful, short name. This is the string that test developers use to specify the argument value.
This argument is required	Check the box to require that the test developer must specify the argument when adding the QuickCall step. In the popup help for the QuickCall, the text “**(required)**” will appear next to the argument name. (The popup help appears when the test developer types a hyphen “-” while adding arguments.) Default: unchecked

Default value	Optional. Provide a single value that the QuickCall will use for the argument if the user does not specify a value for the argument.
Description	Optional. Type one or more lines of text to completely describe the argument. This text will appear in popup help for the QuickCall when the test developer types Ctrl + Space. **Note** You may define validation rules and properties of commands in the QuickCall argument description as per the syntax shown below. **<Procedure argument description> -- <validation rules and properties separate by semicolon>** When exporting quickcall library, commands for the new custom session type will be developed as per the validation rules and properties defined.See Chapter 34, "Session Builder"

Arguments page toolbar

Add	Add a new named argument definition.
Remove	Delete the selected argument definition.
Move Up / Move Down	Move the selected argument definition up or down in the list. When a test developer adds a QuickCall step, the help text displays the arguments in the listed order.

Editing a property setting in multiple QuickCalls

To edit a property setting in multiple QuickCalls, select the QuickCalls (Ctrl-click for multiple) and then change the setting; the setting applies to all selected QuickCalls.

Undo/redo (Ctrl+Z, Ctrl+Y) apply to any change.

QuickCall 'Best practices'

- Do ***not*** use the **open** action (open a session) in a QuickCall.
- When designing a procedure that should span multiple sessions (for example open a session on multiple devices or multiple sessions on a device), define a procedure. QuickCalls are inherently limited to procedures that are appropriate for the current session. Use QuickCalls only for procedures that should happen in a single session.
- Use a naming convention for QuickCalls and add a list of appropriate keywords as the last line in the **Definition** text for QuickCalls. During manual testing, the `Execute a QuickCall` wizard displays only QuickCalls that are appropriate for the current session. In addition, the manual tester can use a keyword to filter the list of QuickCalls even further. As a result, they can quickly find the appropriate QuickCall from what might be a very long list, simplifying their job.
- You define a QuickCall library for use by a broad class of devices. For particular device in the class, you can override a particular QuickCall definition by defining it in a session profile that inherits the base session profile. For example, a QuickCall named **createNewUser** in the base session profile is overwritten by a QuickCall named identically in the QuickCall library used by the inheriting session profile.

Note There is a potential confusion when a **createNewUser** step appears in the test report; you might ask "Which one ran, the one in the base or the other one?". In this case, select the step and look in the Query view for the name of the QuickCall library that the QuickCall is defined in.

- Within a QuickCall library, you might design QuickCalls for use only within the library — utilities for use by other QuickCalls that are defined in the library. The typical QuickCall is meant to be public, that is, to be used by test developers and manual testers in actual tests.

Executing a QuickCall during a manual (interactive) session

Executing a QuickCall during an interactive session

During an interactive session, iTest executes each step in the QuickCall as if you had typed each command yourself. So, with a single click (typically), you can interactively perform a complex initialization or clean-up routine or submit a long sequence of steps that must happen together.

When you execute a QuickCall, iTest captures the action in the Capture view.

To execute a QuickCall during a manual session

1 You can either click QuickCall or click the down-arrow on the button.

- Click QuickCall to open the **Execute a QuickCall** wizard (described in the next step).
- Click the down-arrow to select a QuickCall from a drop-down list.

Tip **Ctrl+Shift+Q** is the same as clicking QuickCall.

2 Now:

- If the selected QuickCall does not use named arguments, iTest immediately executes the steps in the QuickCall. (This is the default behavior — you can set a iTest preference that causes iTest to display the **Execute a QuickCall** wizard [described next].)
- If the QuickCall uses named arguments or if you clicked QuickCall, iTest opens the **Execute a QuickCall** wizard to enable you to view named arguments and to modify argument values as needed. The result of your work in the wizard will be to execute the QuickCall using the argument values you specify.

Tips The **Execute a QuickCall** wizard is optimized for keystroke-only use so that manual testers need never use the mouse while performing a QuickCall.
You can use the wizard to execute QuickCalls in any active session, not just the session that you used to start the wizard.

Any active session whose session profile refers to a QuickCall library appears in the list. There are four active sessions with QuickCalls in this example.

When we expand the **T-2000** session, we find that it can execute two different QuickCalls: **config_for_stress** and **login**.

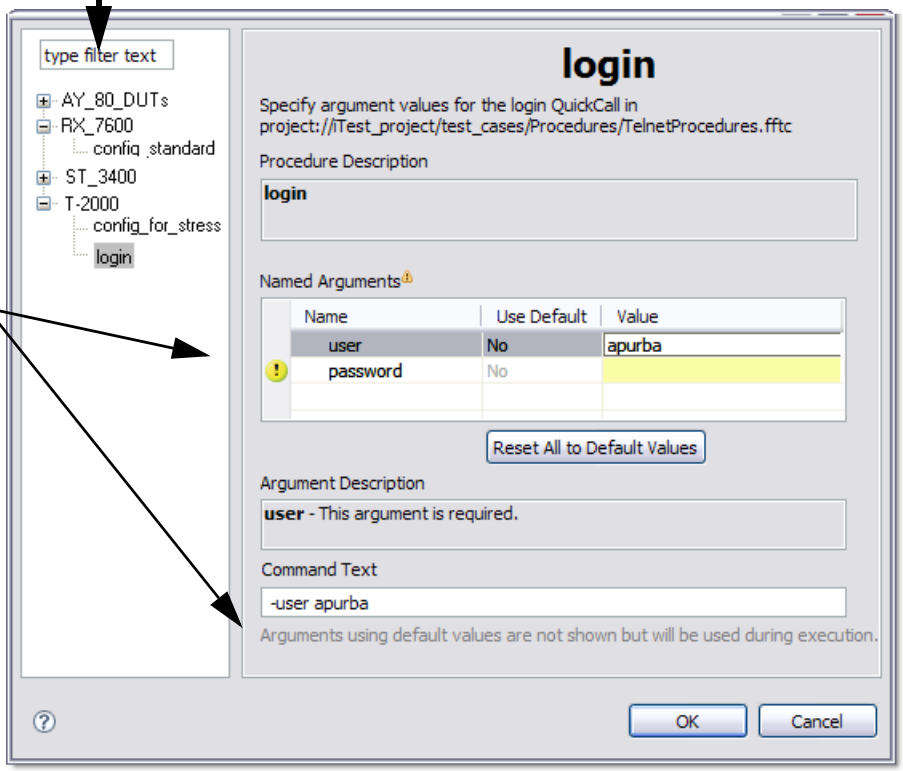
In this example, we are specifying argument values for the QuickCall named **login**.

Use the **filter** text box to limit which QuickCalls appear in the list.

Any changes that you make to the argument values in the **Named Arguments** section are reflected in the **Command Text** box.

When you select an argument, the **Argument Description** box displays information from the definition of the argument.

The text that appears in the **Command Text** box will be executed in the session when you click **OK**. You can edit the text if needed (for example, to modify an argument value or to add numbered arguments).



3 Now you can view and edit argument settings.

As you work, you will notice that the **Command text** box is updated to reflect your changes. The text in the **Command text** box is usually exactly what you need, but if necessary, you can edit the text directly (for example, to modify an argument value or to add numbered arguments).

Note The text in the **Command Text** box follows the normal convention for procedure calls; an argument that uses its default value does not appear. When the **call** step executes, however, the default value will be passed to the QuickCall in the normal way.

- When you select an argument, the **Description** box displays the argument name and other property settings for the argument. If the person who defined the argument set the appropriate values, then this information should help you to understand the argument and its usage.
- The **Arguments** table lists all named arguments. An argument's default value (if defined) appears in the **Value** cell.
  - To change a value, click in the **Value** cell and type the new value.

- To reset a particular argument to its default value, change the **Use Default** setting to **Yes**.
- To set each argument to its default value, click **Reset All to Default Values**.

- The ⚠ icon appears in the title of the **Arguments** table and in the first column for any required argument that currently has no value.
- The wizard does not add numbered arguments. You can type numbered argument values into the **Command Text** box after all of the named arguments. For details on specifying argument values, see “About arguments in QuickCall steps” on page 216.

Tip By default, arguments support runtime substitution of field replacement text. To disable substitution for an argument value, wrap the value inside { and } brackets. As a result, the argument text will be passed exactly as it appears and no substitution will occur.

4 When you finish working in the wizard, click **OK**. iTest executes the QuickCall using the argument values that you specified.

◆ **To insert comments or ‘sleep’ steps while capturing manual tests**

While working in an interactive session that you plan to save as an automated test case, you can type a comment—the text will later appear in the captured test case as a **comment** step.

In the same way, you can insert a **sleep** step at the appropriate location in the resulting test case.

To insert a **comment** or **sleep** step, click the down-arrow in the **QuickCall** menu and select the appropriate action.

Adding a test case step that executes a QuickCall

Executing a QuickCall in a test case

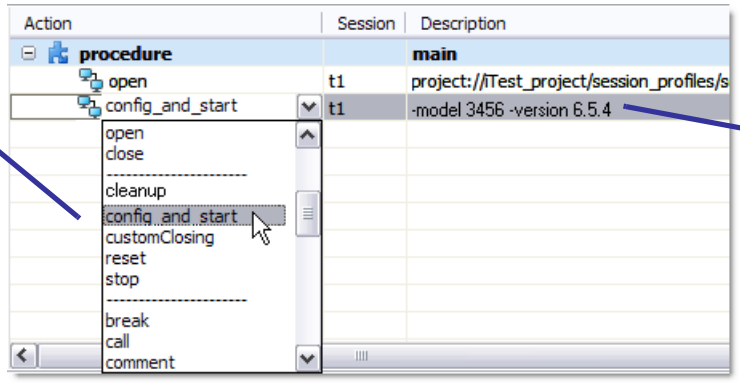
◆ **To add a a step that executes a QuickCall**

Adding a QuickCall step to a test case is exactly like adding any other step:

1 The **open** step for the session must refer to a testbed device or session profile that is associated with a QuickCall library. See “First, specify the QuickCall library to associate with a device or session profile” on page 207 for details.

2 Add a step. Click in the **Action** cell and then select the QuickCall from the list of actions. The QuickCall group appears after the **open**/**close** actions and before the EXEC actions.

These are the QuickCalls that are defined in the QuickCall library referenced by the session profile for session **t1**.



iTest suggests argument values. This information comes from the definition of the QuickCall.

3 The **Description** cell is populated with any arguments and argument values that are defined for the QuickCall. For details, see “About arguments in QuickCall steps” on page 216.

About arguments in QuickCall steps

A QuickCall has the following content in the **Description** cell (all items are separated by spaces).

- Optional: Any number of space-separated ***named arguments*** in the following format:

  `-arg_1 value1 -arg_2 value2 -arg_3 value3 ...`

  You can specify a value dynamically using either a field replacement or **$varName**

  Note All named arguments must appear before all numbered arguments.

- Optional: Any number of values for ***numbered arguments***

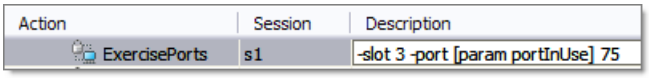
  To access the value of a numbered argument, use **${arg[<number>]}**, for example, **${arg[3]}**

Example:

This example call to the **ExercisePorts** QuickCall includes two named arguments and one numbered argument. Here is the form of the call:

<QuickCallName> -slot *slotNumber* **-port** *portNumber numberOfRepetitions*

Here is the actual QuickCall step: The value of the **port** argument is determined dynamically by the return value of a **param** command.The numbered argument has the value **75**.



Specifying an initialization QuickCall that should execute immediately when a session starts

In the definition of any testbed device or session profile that refers to a QuickCall library, you can configure one of the QuickCalls in the library to execute immediately after the session starts (before any other step).

When you open a session manually (by double-clicking the session profile) the initialization QuickCall executes as soon as the connection to the session is made. The QuickCall is captured as a single step. If you save the captured session to a test case, the initialization QuickCall appears immediately after the **open** step for the session.

To specify an initialization QuickCall

1. From the testbed device or session profile that the QuickCall library is associated with:
   - In the **Device Properties** section on the Testbed editor, open the **Libraries** page.

     or
   - On the Session Profile editor, open the **Misc** page.
2. In the **QuickCall** section, select the QuickCall from the drop-down list for the **Initialization QuickCall** property.

CAUTION Do not use the **open** action (open a session) in an initialization QuickCall (or in any QuickCall).

Stopping execution of the current QuickCall: The ‘return’ action

An **return** step inside the definition of a QuickCall immediately stops executing the current QuickCall and continues execution after the step that called the QuickCall. Any threads started by the QuickCall continue.

An appropriate execution message appears in the Execution view, in the Step Issues view, and in test reports.

A typical use is to place the **return** step in an **if-then** construct that branches based on the response from a session.

The **return** action has no configurable properties. See “The ‘return’ action: Returning execution from the current procedure” on page 234.

Adding text into the response of a QuickCall step: The write action

A **write** step adds text into the response of a QuickCall step. In a QuickCall, you can use one or multiple **write** steps to return a response that includes response data from one or more of the QuickCall’s steps.

Note A **write** step does not write to files or involve file I/O.

See “The ‘write’ action: Adding text into the response of a call step” on page 235.

Viewing a QuickCall definition while working on a QuickCall step

While working in a test case, you might want to review the details of a QuickCall. For any step that executes a QuickCall that is defined in another test case (a QuickCall library), right-click

the QuickCall name and select **Open**. The QuickCall library opens in the Test Case editor to the referenced **QuickCall** definition.

Tip After iTest executes a QuickCall, the Query view lists the QuickCall library where the QuickCall is defined.

Using a QuickCall to open a connection through a terminal server

Using a QuickCall to open a connection to a terminal server is common because the QuickCall can accommodate the many possible states that a serial connection may have been left in (connected, logged in, logged out, and so on). Follow these guidelines:

- Open sessions locally (not in the definition of the QuickCall).
- Call an initialization QuickCall that does all the setup work.As a result, the calling test case has knowledge of the session type. Knowledge of session types is important for activities like right-click > Insert/Parameter, so you can specify session parameters.

Setting QuickCall preferences

QuickCall preference settings

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Sessions**.

Display a warning when executing a QuickCall with missing required arguments	During an interactive session, when you execute a QuickCall that requires a value for an argument, but you did not specify a value, iTest displays a warning dialog box. Uncheck the box to eliminate such warnings. Default: checked
Always open the 'Execute a QuickCall' wizard when executing a QuickCall manually	During an interactive session, when you execute a QuickCall that requires values for named arguments, iTest opens the **Execute a QuickCall** wizard to help you to specify values. To cause iTest to open the wizard for all QuickCalls (whether the QuickCall uses named arguments or not), check the box. Default: unchecked

CHAPTER 10

Procedures

Overview: Procedures in test cases

A procedure is a set of steps that performs an identifiable operation like initializing a device or starting and stopping traffic on a particular port. You **call** the procedure from a test case to perform its operation. For example, the **setupRouter** procedure might include:

- Steps that reset the routing table and port assignments
- Parameters with settings that specify the firmware on the device (the parameter values will be used to adjust the syntax of the steps because, for example, the syntax changed in the latest firmware revision)
- A return string of either "**success**" or "**fail**" to tell the test case (which called the procedure) whether the router was successfully set up and therefore how to proceed
- A **return** or **write** step in the procedure to return a value to the caller. The returned value becomes the response to the **call** step in the caller. Now you can extract the data using a response map, or simply store the data in a variable and use it in the calling procedure.

Procedures are powerful because, when you update a procedure, then every test case that calls the procedure is updated as well. Procedures make test steps reusable.

Test cases are made up of one or more named procedures (by default, each test case has one procedure named **main**). You have the option to create any number of procedures within a test case. Any step in a test case can call a procedure that is defined in the current test case or in other test cases. In addition, you can create a special test case to act as a procedure library — other test cases can call any of the procedures in the library.

Tip You can associate a response map with a procedure so that whatever is returned by the procedure will automatically have "blue boxes" — structured queries available on the corresponding **call** step.

Technical description of a procedure

So, a procedure technically consists of:

- A sequence of steps
- Optionally, a set of input parameters that customize the steps
- Optionally, an output value (called a *return value*)

Why create procedures?

The most common use for procedures is for tasks that you want to execute often, for example, resetting ports, initializing a device (before starting a test or to "clean up" after execution), setting up a particular configuration in preparation for checking a particular value, and so on. In any test case, rather than copy/pasting all the steps that perform the operation, you add a single **call** step that executes the procedure.

The benefits? The test case is more modular and therefore more easily …

- **Understood**: Test case developers can read the procedure name (for example, **setupRouter**) and understand what's happening without having to see all the details.
- **Debugged**: Because the functionality in the procedure is isolated, once you know that the procedure is working as expected, you can ignore the procedure and focus on other problem areas.
- **Ported**: When a set of tests has to cover a new revision, just update the procedure as needed and every test case is that much further along.
- **Maintained**: When you update a procedure, all test cases that use the procedure are automatically updated too.

Local procedures, foreign procedures, and procedure libraries

A ***local*** procedure is a procedure that is defined in the same test case from which it is called — defined in test case **A** and called from test case **A**.

A ***foreign*** procedure is a procedure that is defined in a different test case than the one from which it is called — defined in test case **A** and called from test case **B**.

There are two important uses for foreign procedures:

- You might want to build up a library of procedures that can be used by any test case. For this purpose, you can create a test case (named, for example, **myProcedureLibrary**) with a blank main procedure and multiple defined procedures. See "Creating a procedure library" on page 237.
- Your test case might call procedures in a variety of other test cases or from a test case that acts as a procedure library. (This differs from executing the main procedure of a referenced test case using the **run** action).

Using QuickCalls

You should know about a powerful alternative to procedures: The iTest QuickCall feature makes it easy to add custom actions like the built-in iTest actions (built-in actions, for example, **getTable** or **walk** in SNMP sessions). You can use QuickCalls both during interactive (manual) testing and as test case steps. QuickCall procedures are custom actions that ***you*** define for a session profile or device. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

How procedures execute

- When a **call** step or **CallProcedure** action executes, it calls a procedure. The called procedure becomes the currently executing procedure. Normal execution flow continues in the procedure and, when the procedure exits, the caller again becomes the currently executing procedure.

- The steps in local procedures have the same execution context as the **main** procedure. (See "Local procedures, foreign procedures, and procedure libraries" on page 220.)

When a **call** step executes, all steps in the called procedure execute in the same thread (unless steps in the procedure are specified as asynch).

- iTest does not enforce timeout settings for **call** steps.
- The currently executing procedure waits for all outstanding executing steps and analysis rules to complete before executing a **call** step or **CallProcedure** action.
- Sessions span procedural contexts. For example, a CLI session opened in a caller is still active for steps in a called procedure (the **Session** name must be the same).
- Parameter values are resolved from parameter property settings in the following precedence order: First, the calling test case document, then, the current foreign test case document, and last, the session profile associated with the calling step. See "Parameters" on page 222 for details.
- The **Procedure** column in the Execution view identifies the procedure name associated with each step.
- A called procedure can use a **return** or **write** step to populate the response of the calling step. The calling step's analysis rule applies to the result in the normal way.
- In any procedure, the **AbortExecution** event action terminates the entire test case and sets test result to **Abort** .
- **Pass**, **Warning**, **Abort**, and **Fail** settings apply to the overall test result regardless of which procedure declared them.

Note **Advanced Users:** The default entry point to a test case is the procedure named **main**. You can rename **main**, but to ensure that the test case starts properly, change the **Entry point** setting (on the Test Case editor **General** page) to the new name.

Tip **Invoking a TCL routine**

Insert a step that opens a Tcl Shell session. You can execute any TCL commands in that session, including sourcing and executing scripts.

Advanced Users: About procedures

Note The **Test Case Editor** displays an error or a warning as set in "Setting preferences for the Test Case Editor" on page 190 (Chapter 10, "Procedures"), under these circumstances:

- Default value has been specified in the **Procedure** arguments, and
- Argument values are not specified in the calling **Step**.

How iTest passes arguments to procedures

iTest passes arguments to procedures "by value", not "by reference". That means that if you pass an argument to a procedure and the procedure modifies the data, the caller will not have access to the changes.

There are three ways to accomplish passing information back to calling procedure:

- Use a **return** or **write** step in the procedure to return a value to the caller. The returned value becomes the response to the **call** step in the caller. Now you can extract the data using a response map, or simply store the data in a variable and use it in the calling procedure. (Remember that each variable is local to its procedure.)
- Use global variables. **Not recommended**.
- iTest does not have **uplevel** or **upvar** like Tcl does, but if you understand iTest's variable positioning in the heap, you can modify variables in the parent scope from the called procedure. For iTest, a variable is an XPath query used to extract data from the heap (which is an XML document). The root from which iTest searches this variable depends on the scope. If you are inside a procedure, iTest will search for this XPath query from certain node in the tree. This allows iTest to implement variable scoping.

  iTest stores all global variables under the **/data** node. So if you do **gset i 0**, you will see a node in the Data view: **/data/i** whose value is zero.

  iTest stores procedure variables at each appropriate stack level. You can see stack nodes that are organized like: **/data/stacks/stack/frame/frame/variable** for the first procedure, **/data/stacks/stack/frame/frame/frame/variable** for the second procedure. You can use this information to do equivalent of Tcl **uplevel**. To modify a variable in the calling procedure from the called procedure, do **set {../variable_in_calling_proc} 5**.

The key is that all iTest variables are a shortcut way to represent access to data in an XML tree. This provides a lot of power, but should be used cautiously making sure that readability and debugability of the test cases do not suffer.

Parameters

Parameters are distinct from variables. Parameters are loaded from parameter files, test cases, testbeds, or session profiles and are "read-only". Parameters are available to any procedure at any level. One does not have to pass parameters to a procedure.

While parameter values are accessible using the **param** command, the command does not allow you to set values. Even something like **call myProc -arg1 [param myParam]** is not used, as iTest is only passing the value — you are not touching the original parameter. In addition, there is no need to pass parameters to a procedure unless you intend the procedure to take a variable value as well, because the procedure can access the parameter **myParam** using **[param myParam]** in its context.

Parameters in procedures

Parameters are loaded in by **call** steps, so a parameter's resolved value depends on the order in which procedures are called. For example, two procedures use the same parameters with different values: If you call a procedure from test case **B** and then call a procedure from test case **A**, then the parameter values for **B** are resolved first. If **A** is called before **B**, then **A**'s values are resolved first.

Note For QuickCalls, parameters are resolved differently. See "How QuickCalls execute" on page 206.

Called procedure argument validation

Many languages validate arguments passed to procedures. But many scripting languages do not — you discover any errors at runtime. iTest does not validate any of the arguments passed to a

call procedure statement. iTest is designed for procedures to have very loose, variable arguments syntax. Typical **call** syntax looks like:

call procName -myArg mouse -myOtherArg rabbit cat dog

You can see that we are passing two named arguments and two unnamed arguments (**cat** and **dog**). All of the arguments are optional. The arguments become stack variables in the context of the called procedure. If you do not call the procedure with the correct arguments, your variable access in the called procedure will produce a run-time error.

Use the **Procedure Call** wizard to help you to avoid issue of not finding errors until run-time. See "Creating a 'call' step using the Procedure Call wizard" on page 229.

Defining a procedure

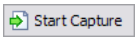
Each procedure must have a unique name.

Important Do not name a procedure "**procedure**" or "**Procedure**".

You can copy and paste procedures from one test case to another.

To define a procedure

You can add a procedure definition to a test case in either of the following ways:

- Manually execute the steps that you want to include in the procedure and then, using the capture-to-test feature 

or the Capture view, save the captured session as a procedure in a test case. The procedure is added after the last step in the test case. This option is described in "Adding captured steps into a test case or Python Script" on page 100.
- While working in the Test Case editor, add a procedure "manually".

This topic provides instructions for adding a procedure manually and for modifying procedure properties (like arguments).

Tip On the Test Case editor **Steps** page, click **Collapse All** to view only the procedure names and not the individual steps.

Adding a procedure definition manually and modifying procedure properties (like arguments)

1 In the Test Case editor, select a step and click **Insert**, and then select **Insert Procedure**. iTest adds a blank procedure definition below the selected step.

Select the **procedure** label to display the **Procedure Properties** tree at the bottom with the following branch structure:

General, Description, Inputs and Outputs with sub-branches, Arguments and Response.

Note

While working on a test case in the Test Case editor, right-click and on the menu displayed, select (**Show Properties View**). The Properties view appears as a tabbed pane on the iTest window.

The **Properties View** pane display is context-specific and varies depending on where your cursor is positioned. You may edit step properties using either the **Step Properties** section (within the Test Case Editor) or via the **Properties View** tab.

For example, if your cursor is at the **procedure** label (**Steps > Action> procedure**) and then select from the right-click menu, the **Properties** pane opens to display **Procedure Properties** (which is also displayed below the **Steps > Actions** section).

Note You may position and resize the tabbed Properties view window as required.

2 **General** page: In the **Procedure Properties** tree, click **General**. On the **General** page, specify values for the following properties. The text that you provide for the **Name**, **Headline**, **Author**, and **Version** properties (along with the **Description** property specified on the **Description** page) are displayed on the popup help for the procedure. (The popup help appears when the test developer types Ctrl+Space while in the **Description** cell.)

Name	Required. Provide a meaningful name for the procedure. This is the string that test developers use to call the procedure. For procedures: This string will appear in the drop-down list of procedures in the **Description** cell for **call** steps or **CallProcedure** actions. For QuickCalls: Do not use the name of an existing iTest action. If you do, then during execution, iTest generates an Execution Issue Warning and then executes the built-in action.
Include this procedure when listing callable procedures	Check the box to include this procedure's name in the drop-down list of procedures in the **Description** cell for **call** steps or **CallProcedure** actions. This option makes the procedure available for use in any test case in the current workspace. Default: Unchecked
Headline	Optional. Type a single line of text that describes the procedure. This string will appear with the procedure name in the drop-down list of procedures in the **Description** cell for **call** steps or **CallProcedure** actions. This text also appears in the **Headline** column of the Favorites view to help you when selecting a procedure.
Author	Optional. Type the name of the person who created the procedure definition. This helps coworkers who have questions about the procedure.

Version	Optional. If you track changes to the procedure definition, use this property to specify the version number of the procedure.
Response Map	Optional. Specify the response map to apply to the value returned by the procedure (via a **return** step in the procedure). **Note** Spirent recommends that you specify the response map *here*, rather than in the **call** step in the caller. (The **call** step does not need to have a response map configured for it.) In this situation, consider defining a QuickCall to perform the procedure's function. Because a QuickCall is associated with a session profile, you can associate the required response map with the session profile and do not need to worry about response maps while defining test case steps. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".
Default session	Optional. Specify a session type in the following situation: If the Session ID of the steps in the procedure are parameterized so that the Session ID value can be set at runtime, then specify the appropriate session type here. This enables the system (at design-time, while you are designing the procedure) to populate the **Action** cell with actions that are appropriate for the session type.

3 **Description:** In the **Procedure Properties** tree, click **Description**. For the **Description** property, type the text that should appear in the help popup for the procedure. Spirent recommends that you show all forms of the procedure call (for example, provide examples of all permitted combinations of required and optional arguments). Show each form of the call on a new line.

4 **Inputs and Outputs:** Procedures can use both named arguments and numbered arguments. Perform this step if the procedure uses named arguments. The order of arguments in the list that you create here is the order in which they appear when a test case developer is creating a **call** step or **CallProcedure** action.

a **Arguments:** In the **Procedure Properties> Inputs and Outputs**tree, click **Arguments**. On the **Arguments** page, click to add a new argument definition.

Specify values for the following properties for arguments to the procedure:

Name	Provide a meaningful, short name. This is the string that test developers use to specify the argument value.
This argument is required	Check the box to require that the test developer must specify the argument when calling the procedure. In the popup help for the procedure, the text **(required)** will appear next to the argument name. (The popup help appears when the test developer types a hyphen "-" while adding arguments.) Default: Unchecked
Default value	Optional. Provide a single value that the procedure will use for the argument if the user does not specify a value for the argument.]
Description	Optional. Type one or more lines of text to completely describe the argument. This text will appear in popup help for the procedure when the test developer types Ctrl + Space.

Arguments page toolbar

Add	Add a new named argument definition.
Remove	Delete the selected argument definition.
Move Up / Move Down	Move the selected argument definition up or down in the list. When a test developer adds a **call** step or **CallProcedure** action, the help text that appears for the procedure displays the arguments in the listed order.

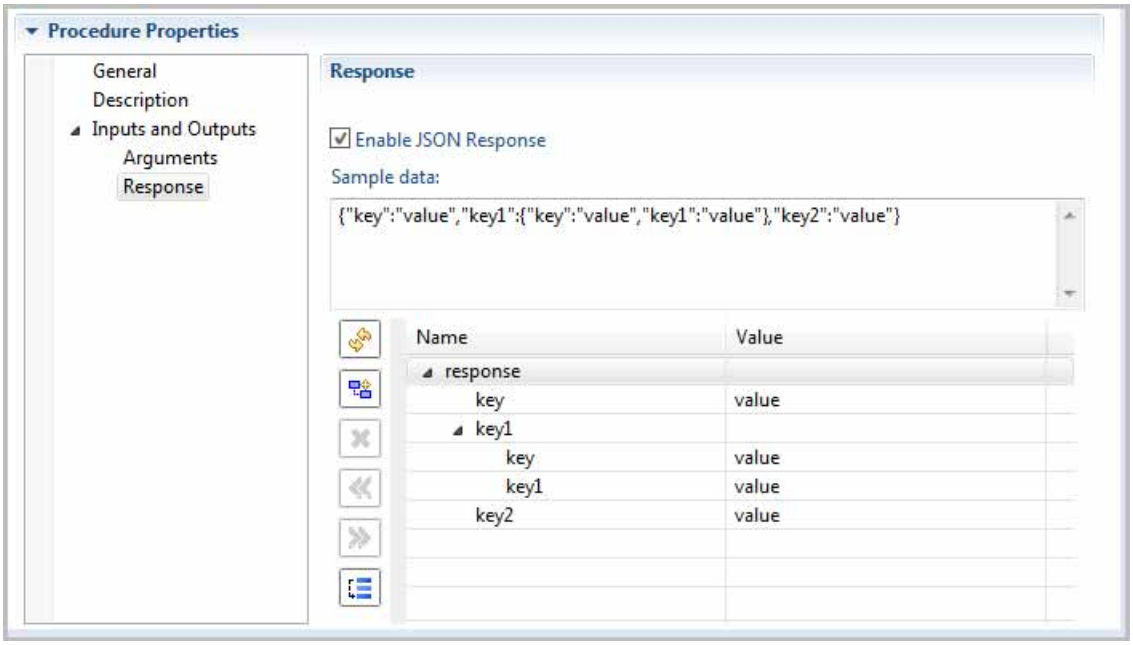
b **Response:** In the **Procedure Properties> Inputs and Output** tree, click **Response.** On the **Response** page, select **Enable JSON Response** to configure the sample JSON response. Use the text area to edit the JSON string, or use the JSON tree to generate the JSON string.

Enable JSON Response	Select to enable JSON Response, which allows you to define the JSON response. When enabled, the response node in the **Procedure Properties > Inputs and Outputs** tree allows you to configure the JSON response for this procedure. When disabled, the sample JSON structure controls (raw text and nested/indented layout) is not available (grayed out).

You can also get a sample data from response map, if you Enable JSON Response and the sample JSON response is empty (not defined yet). The first sample data from the response map file will be fetched and populated on the Response page.

You may define QuickCall procedures with JSON Response, and insert these calls in a Test Case. The insterted step populates the **Response View** with the JSON response from the called procedure (the Response View background is light grey before running a test case). It is not necessary to execute the test case to see the format of the response.

Note The response map should use JSON mapper for the data to be populated in the Response window.



Response toolbar

Refresh 	The modification on JSON tree will be updated to the sample data text area automatically, while the modification on sample data text area does not update to JSON tree structure. Use this button to refresh the JSON tree.
Add Child 	Add a new JSON data element under the selected element as a child in the response structure. The default JSON response is populated with {“key”:"value"} (as shown above). For example, select the **response** key in the structure and click **Add Child** to add a new key/value pair at the root of the JSON structure, resulting in {"key":"value","key1":"value"}.
Delete 	Delete the selected JSON element.
Decrease / Indent 	Decrease / increase indent to change the JSON data structure. Decrease to move the element up one level. Indent to move the element down one level.
Format 	Toggle to change the format of the JSON data text.

Insert Return Value Dialog

Use the **Insert Return Value dialog** to get the **XPath** for sample json response. See "Commands that return sample json xpath used in Store Processor" on page 384. Once the sample json response (Step Note , page 227) is defined for the procedure, the tree style of that json string displays in this dialog, which is the same as the json tree in the Response Structure (see The response map should use JSON mapper for the data to be populated in the Response window.The response map should use JSON mapper for the data to be populated in the Response window.The response map should use JSON mapper for the data to be populated in the Response window.page 227 page) under the **Procedure Properties> Inputs and Outputs > Response** window (Step Note , page 227).

To get the **XPath** for the node value, select the node in the tree, the corresponding field replacement will be generated in this dialog. After clicking the Insert button, the [**return query_xpath**] will be inserted automatically.

Editing a property setting in multiple procedures

To edit a property setting in multiple procedures, select the procedures and then change the setting; the setting applies to all selected procedures.

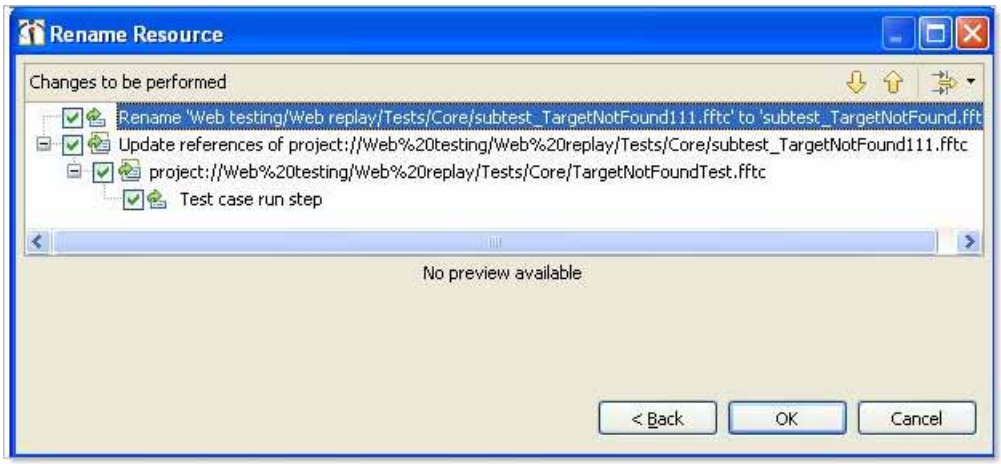
Undo/redo (Ctrl+Z, Ctrl+Y) apply to any change.

Calling the procedure

See "Calling a procedure in a test case step or in a property setting" on page 229

Renaming or deleting a procedure

As you can imagine, renaming or deleting a procedure definition could have serious consequences (a test case would not be able to find a procedure that is supposed to execute). To help you to rename or delete procedures safely, iTest presents a dialog box whenever you make such a change:



When you click **OK**, iTest auto-updates all selected procedure references to the new name — the changed procedure will function exactly as before the change.

You must edit references that you elected not to auto-update individually. You might want to print the list so you can be sure to edit each of them.

Calling a procedure in a test case step or in a property setting

Note Not applicable for Python test cases.

You can use a **call** step or **CallProcedure** action to execute a local or foreign procedure. You cannot use the **run** action to execute a procedure. The **call** action is defined in "The 'call' action: Calling a procedure" on page 233.

For **call** steps or **CallProcedure** actions, the **Command** cell contains the target procedure name and any arguments and argument values.

Procedures can call procedures in a nested fashion.

Procedure call syntax

procedureName [*-namedArg1 arg1Value -namedArg2 arg2Value* ...] [*numberedArg1 numberedArg2* ...]

- Arguments are optional
- Any named arguments must appear before any numbered arguments

Example

This example call to the **ExercisePorts** procedure includes two named arguments and one numbered argument. Here is the form of the procedure call:

ExercisePorts -slot *slotNumber* **-port** *portNumber numberOfRepetitions*

Here is the actual text that appears in the **Description** cell for the **call** step (the value of the **port** argument is determined dynamically by the return value of a **param** command).

```
ExercisePorts -slot 3 -port [param portInUse] 75
```

For details, see "About arguments in procedure calls" on page 234.

To call a procedure

Use one of the following tools:

- Recommended: Use the **Procedure Call** wizard. See "Creating a 'call' step using the Procedure Call wizard" on page 229.
- Use the in-line editing tools to define the **call** command. See "Creating a procedure 'call' step using in-line editing" on page 231.
- Right-click a foreign procedure in the Favorites view. See "Creating a procedure 'call' step from the Favorites view" on page 233.

Creating a 'call' step using the Procedure Call wizard

You can use the **Procedure Call** wizard to quickly define a **call** step or **callProcedure** action. The single-page wizard displays everything there is to know about a procedure and its named arguments and helps you to modify argument values as needed. The result of your work in the wizard is to generate the call command text for the **call** step or **callProcedure** action.

The wizard generates normal procedure syntax, as described in "Procedure call syntax" on page 229.

To add a call step

Start the **Procedure Call** wizard using one of the following methods:

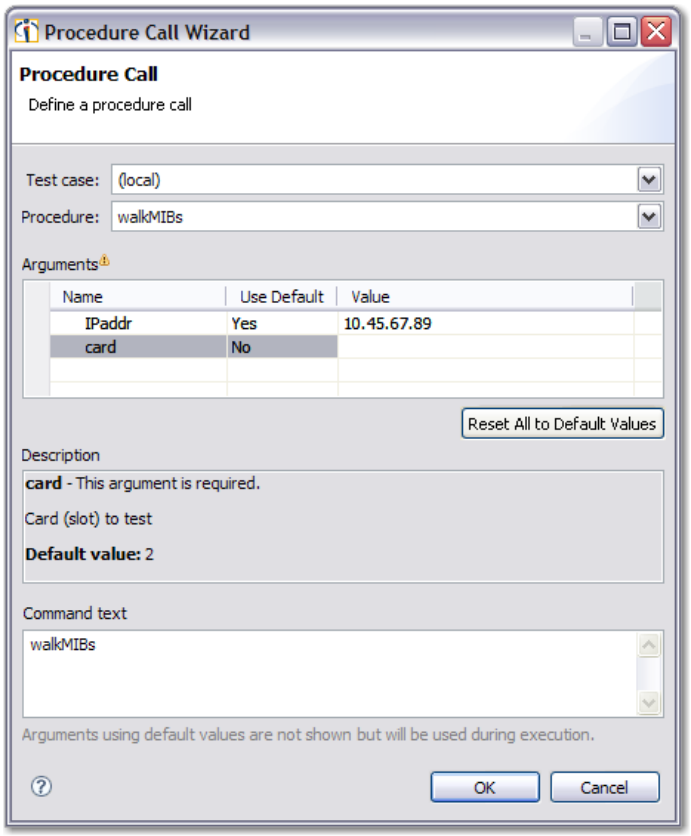
1. Select the step just before where the **call** step should go.
2. Now, do one of the following:
   - In the toolbar, select the arrow on the Insert button and select **Insert Procedure Call Using Wizard**
   - Right-click the step and select **Insert > Insert Procedure Call Using Wizard**
   - In the **Test Case** menu, select **Insert > Insert Procedure Call Using Wizard**
   - Add a step with an **Action** of **call**. Optional: If you know the name of the procedure to call, you can specify its name in the **Description** cell (as described in "Step 2: Specify the procedure to call" on page 232). As a result, the wizard page will be "filled in" with information about the procedure. Click ... in the **Description** cell to start the **Procedure Call** wizard.

   Now, work in the Procedure Call wizard

Any changes that you make in the upper sections of the page are reflected in the **Command text** box.

When you select a test case, procedure, or argument, the **Description** box displays information about the selected item.

The text that appears in the **Command Text** box will appear in the **Description** field for the **call** step. You can edit the text if needed (for example, to add numbered arguments).



3. In the **Test Case** box, select the test case (procedure library) that contains the procedure.
4. In the **Procedure** box, select the procedure.
5. Now you can view and edit argument settings.

   As you work, you will notice that the **Command text** box is updated to reflect your changes. The text in the **Command text** box is usually exactly what you need, but if necessary, you can edit the text directly (for example, to add numbered arguments).

Note The text in the **Command Text** box follows the normal convention for procedure calls; an argument that uses its default value does not appear. When the **call** step executes, however, the default value will be passed to the procedure in the normal way.

- When you select an argument, the **Description** box displays the argument name and other property settings for the argument. If the person who defined the argument set the appropriate values, then this information should help you to understand the argument and its usage.
- The **Arguments** table lists all named arguments. An argument’s default value (if defined) appears in the **Value** cell.
  - To change a value, click in the **Value** cell and type the new value.
  - To reset a particular argument to its default value, change the **Use Default** setting to **Yes**.
  - To set each argument to its default value, click **Reset All to Default Values**.
- The  icon appears in the title of the **Arguments** table and in the first column for any required argument that has no value.
- The wizard does not add numbered arguments. You can type numbered argument values into the **Command Text** box after all of the named arguments. For details on specifying argument values, see “About arguments in procedure calls” on page 234.

Tip By default, procedure arguments support runtime substitution of field replacements. To disable substitution for an argument value, wrap the value inside { and } brackets. As a result, the argument text will be passed exactly as it appears and no substitution will occur.

6 When you finish defining the procedure call, click **OK**. iTest places the text from the **Command Text** box into the **Description** cell for the **call** step.

Creating a procedure ‘call’ step using in-line editing

Procedure call syntax

procedureName [*-namedArg1 arg1Value -namedArg2 arg2Value* ...] [*numberedArg1 numberedArg2* ...]

Example

This example call to the **ExercisePorts** procedure includes two named arguments and one numbered argument. Here is the form of the procedure call:

procedureName -slot *slotNumber* **-port** *portNumber numberOfRepetitions*

Here is the actual text that appears in the **Description** cell for the **call** step (the value of the **port** argument is determined dynamically by the return value of a **param** command).

```
ExercisePorts -slot 3 -port [param portInUse] 75
```

For details, see “About arguments in procedure calls” on page 234.

To call a procedure

Step 1: Add the call

1 Add a step and set the **Action** to **call**. Do not specify a **Session** ID.

Tip These instructions also apply if you are adding a **CallProcedure** action to an analysis rule processor. These instructions, however, do not work while you are adding a **CallProcedure** action to an Event definition. The fastest way to add a **CallProcedure** action to an Event, therefore, is to follow these instructions and then copy the text from the **Description** cell into the **Procedure name and arguments** property for the **CallProcedure** action.

Step 2: Specify the procedure to call

Local procedure:

In the **Description** cell, type or select the procedure from the drop-down list.

Foreign procedure:

1 In the **Description** cell, open the drop-down list and double-click the procedure library. The list displays local procedures by name and procedure libraries as URIs followed by a ? character.

2 Now you select a particular procedure from the library, as follows:

- Press Ctrl+Space. iTest displays a drop-down list of the procedures defined in the procedure library.

Note Only the procedures specified as “callable” appear in the list (that is, procedures with the **Include this procedure when listing callable procedures** property checked).

- When you select a procedure, another popup displays the properties for the procedure. Double-click the procedure. It is appended to the URI of the procedure library.

Step 3: Specify arguments

If the procedure uses arguments, follow these steps to add them:

Note There is no need to specify arguments that have defined default values. If you do not add the argument to the command line, then iTest executes the procedure using the specified **Default value** for the argument.

---

1 Type a space to separate the argument from the procedure name.

2 Type a hyphen “-” to display the list of named arguments. When you select an argument, then another popup displays the argument name and the other property settings from the **Arguments** property page (if the properties are set).

3 Double-click an argument to add it to the command for the procedure call. To add a value for the argument, type a space followed by the value. You can use field replacements in arguments.

4 Add all required arguments in this way. Separate all arguments and values using a single space.

- **Step 4: Finish**

  After you finish editing the command text in the **Description** cell (adding the procedure name and any arguments) and exit the cell, the full URI to the procedure library will no longer appear — only the test case name, a ? character, and the procedure name plus arguments remain. This makes it easier to read and understand the step. For example,

  ```
  project://my_project/test_cases/my_lib.fftc?login
  ```

  will change to

  ```
  my_lib?login
  ```

  To change the behavior to always display the full URI, specify the preference setting as follows: Click **Window > Preferences**. On the **Preferences** page, click **Spirent > Editors > Test Case Editor > EXEC steps**. Check **Display the full URIs for foreign procedure calls**.

Creating a procedure ‘call’ step from the Favorites view

Use this method when a test case or procedure library appears in the Favorites view. To add a test case that defines the procedure into the Favorites view, drag it from the iTest Explorer.

Note The Favorites view lists only ‘callable’ procedures (that is, procedures with the **Include this procedure when listing callable procedures** property checked).

1. While developing the test case that should call the procedure, select a step. The procedure call will be added after the selected step.
2. In the Favorites view, expand the test case that contains the procedure that you want to call. Right-click the procedure and click **Insert step to call this procedure**. iTest adds the **call** step to the calling test case and then opens it in the Test Case editor.
3. If the procedure uses arguments, follow the instructions for adding arguments described in “Step 3: Specify arguments” on page 232.

The ‘call’ action: Calling a procedure

The EXEC **call** action transfers execution from the current procedure (the caller) to the first step in a different procedure (the called procedure).

- The **call** action can pass arguments to procedures. Named arguments are listed first in **-arg1 arg1Value -arg2 arg2Value** format, followed by the values of numbered arguments in order. See “About arguments in procedure calls” on page 234.
- When the called procedure exits, execution returns to the caller
- Procedures can call procedures in a nested fashion
- **call** creates an executed step in test reports. The step contains any response data, and the calling step's analysis rule applies in the normal way to the returned data.

Tip A **write** step adds text into the response of a **call** step. In a called procedure, you can use **write** steps to include response text from multiple steps in the called procedure (as a multi-line string). The text that appears in the **Description** cell of the **write** step is appended to the response. See “The ‘write’ action: Adding text into the response of a call step” on page 235 for details.

Note If a **call** step in a child test case **B** (begun by a **run** step in a grandparent test case **A**) calls grandchild test case **C**: The **call**ed test case **C** will use the shared session from test case **A** in its **open** step if the **Session ID** in **C** is same as the **Session ID** in **A**. If you do not want to use the shared session, then change the **Session ID** in **C** to be different from the **Session ID** in **A**.

- **To call a procedure (Adding a 'call' step)**

  See "Calling a procedure in a test case step or in a property setting" on page 229

- **To define a procedure**

  See "Defining a procedure" on page 223.

- **How procedures execute**

  See "How procedures execute" on page 220

About arguments in procedure calls

A **call** action has the following content in the **Description** cell (all items are separated by spaces).

- The ***procedure name*** (you can specify the name dynamically using either a variable or a field replacement)
- Optional: Any number of space-separated ***named arguments*** in the following format:

  `-arg_1 value1 -arg_2 value2 -arg_3 value3 ...`

  You can specify a value dynamically using either a field replacement or **$varName**

  Note All named arguments must appear before all numbered arguments.

- Optional: Any number of values for ***numbered arguments***

  To access the value of a numbered argument, use as follows.

  Tcl: ${arg[<number>]}**, for example, **${arg[3]}

  Python: [arg(<number>)], for example, [arg(3)]

Example:

This example call to the **ExercisePorts** procedure includes two named arguments and one numbered argument. Here is the form of the call:

procedureName -slot *slotNumber* **-port** *portNumber numberOfRepetitions*

Here is an actual call: The value of the **port** argument is determined dynamically by the return value of a **param** command.

```
ExercisePorts -slot 3 -port [param portInUse] 75
```

The 'return' action: Returning execution from the current procedure

A **return** step stops executing the current procedure and returns execution from the current procedure to the caller. This means either continue processing the procedure or QuickCall whose **call** step (or **CallProcedure** action) caused the procedure to start, or end test case

execution if the procedure was the initial entry point. Any threads started by the procedure or QuickCall continue.

- The **return** action does not return a response. To return a value for a procedure, the text specified in the **Description** cell (the value of the **Command** property) of the **return** step is appended to the response of the **call** step. The text string can contain field replacements (for example, **[response var_name]**). See “Tips on using ‘write’ and ‘return’ steps to prepare useful response data for called procedures” on page 237.

The **Start this step in a new thread and proceed to the next step** (asynchronous execution) property on a **return** step is ignored.

- Steps nested inside **return** steps are never executed.
- Contrast **return** with **write**. See “The ‘write’ action: Adding text into the response of a call step” on page 235.

Tip You can use a **return** step in the main procedure to exit a test case. Because you do not typically want to return every time the test case runs, you'll probably include the **return** step within an **if** construct.

The ‘write’ action: Adding text into the response of a call step

---

Note The **write** action is supported for TCL only.

A **write** step adds text into the response of a **call** step. In a called procedure, you can use one or multiple **write** steps to return a response that includes response data from one or more of the procedure's steps.

If you include multiple **write** steps in a procedure, then you can easily add a line terminator to each response so that the resulting returned response is a multi-line string. In addition, the text that appears in the **Description** cell of each **write** step is appended to the response. See “Tips on using ‘write’ and ‘return’ steps to prepare useful response data for called procedures” on page 237.

Contrast **write** with “The ‘return’ action: Returning execution from the current procedure” on page 234.

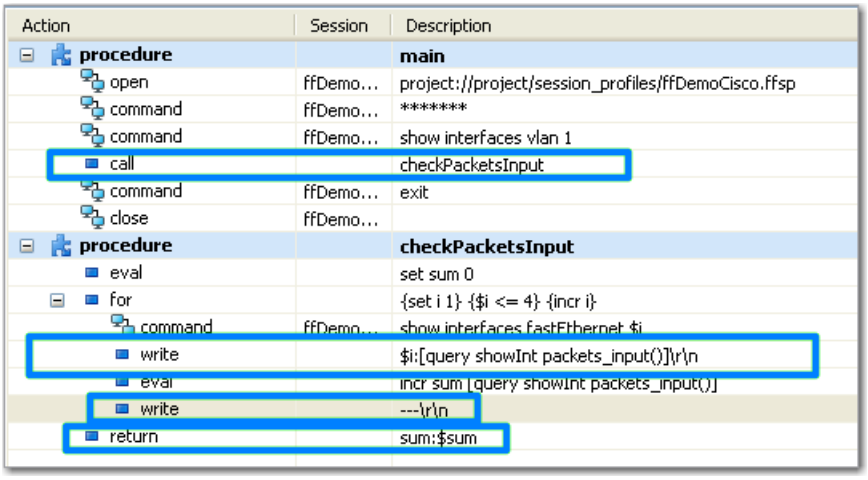
Note A **write** step does not write to files or involve file I/O.

To define a write step

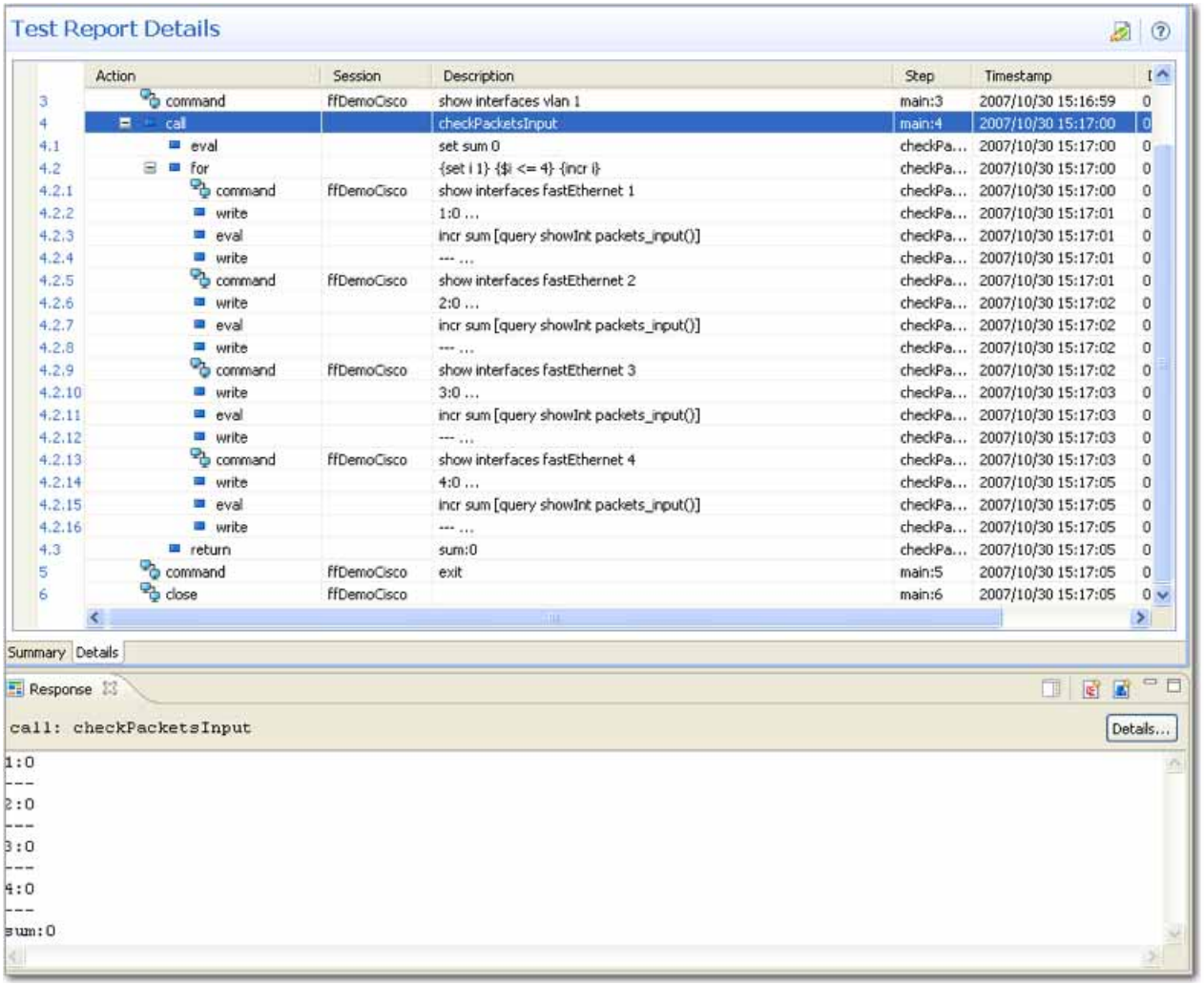
1. In the called procedure, save the information that you want to return into a variable; either:
   - Save the full response into a variable (for later retrieval using a **response** command
   - Use an analysis rule to save a particular part of the data from the response
2. Use a **write** step after the step to display the value (and any clarifying text that you would like to add). See the example. Use **\r\n** to append a line terminator to the **write** step response text.

Example

This example shows **write** steps in the called procedure, the expansion of the procedure in the test report, and the resulting response to the **call** step in the Response view.



The response to the call step includes the content of all of the write steps in the procedure (concatenated into a multi-line string) plus the text in the **Description** cell of the return step.



Tips on using ‘write’ and ‘return’ steps to prepare useful response data for called procedures

If you configure the called procedure with a response map that will parse the response of the **call** step — that is, the content you put in the **write** and **return** steps — then you can easily apply analysis rules to the **call** step.

Response mapping in the procedure

- To make response mapping easier, you can use the **format** action to create more structure to the text that you write into the response. For example, if you are building a table of information to be returned to the caller, you could use something like this to produce a nicely formatted table that is easy to create a response map for:

```
eval       set fmt "%-25s %5d"
write       format $fmt "Description" "Count"
write       format $fmt "-----------" "-----"
eval       set count 0
foreach     val $values
  eval     incr count
  write     format $fmt $val $count
```

- In cases where you want to return several individual values, response mapping can still help. Place each value on a line with the name, followed by a colon, followed by the value. In this format, you can even use automatic response mapping to generate a very powerful map.

Viewing a foreign procedure while working on the call step

While working in a test case, you might want to review the procedure that a step calls. For any step that calls a procedure that is defined in another test case, right-click the procedure name and select **Open**. The other test case opens to the referenced **procedure** step.

Using a procedure to open a connection through a terminal server

Using a procedure to open a connection to a terminal server is common because the procedure can accommodate the many possible states that a serial connection may have been left in (connected, logged in, logged out, and so on). Follow these guidelines:

- Open sessions locally.
- Call a procedure that does all the setup work. Specifically, do ***not*** open sessions in a foreign procedure. This is done so that the calling test case has knowledge of the session type. Knowledge of session types is important for things like right-click > Insert/Parameter, so you can pick session parameters.

Creating a procedure library

It is easy to create a library of procedures that can be shared by iTest users. (For suggestions on why you might create a procedure library, see “Overview: Procedures in test cases” on page 219.)

To create a procedure library, you create definitions of commonly used procedures in a single test case. The test case then acts as a procedure library.

Any iTest test case in the same workspace can then use a **call** step or **CallProcedure** action to call a procedure from the library.

Tip On the Test Case editor **Steps** page, click **Collapse All** to view only the procedures and not the individual steps.

Creating a procedure library

1. Create a new test case.
2. On the Test Case editor's **General** page, check the **Include this test case when listing procedure libraries for call steps and CallProcedure actions** check box. This property identifies the test case as a procedure library that should be made easily available when creating test cases. As a result, when a test case developer adds a **call** step or a **CallProcedure** action to a test case, the URI for this test case appears in the drop-down list of procedure libraries and local and foreign procedures in the **Description** cell.
3. Optional, but recommended: For the procedure named “main”, uncheck the **Include this procedure when listing callable procedures** property (in the **General** section of the **Procedure Properties** page). As a result, when a test case developer adds a **call** step or **CallProcedure** action, **main** does not appear in the drop-down list of available procedures in the **Description** cell.
4. Now add procedures to the test case. See “Defining a procedure” on page 223 for details.
5. Perform this step for each procedure that should appear in the drop-down list foreign procedures in the **Description** cell while a test case developer adds a **call** step or a **CallProcedure** action: Check the **Include this procedure when listing callable procedures** property (in the **General** section of the **Procedure Properties** page).
6. To make the procedures available to other test cases, you must save this test case.

iTest users can now call all of the procedures in this procedure library.

CHAPTER 11

Actions

Actions

As you work in iTest, you will notice that the actions that you take while performing an interactive test (submitting a command, clicking an SNMP MIB or a Web link) become **Actions** in steps when you save the session as a test case. Let's compare how you work with actions while capturing an interactive session and while editing a test case:

Actions in interactive (manual) sessions

You perform actions in a session: send a CLI **command** to a Telnet session, send **get** or **set** actions on a MIB variable in an SNMP session, and so on.

For example, the most common action in Telnet sessions is the **command** action; that is, submit command text to the session. For step **2.4** in the example, the **Action** is **command** and the command text (in the **Description** cell) is **show ip traffic**. The device performs a **show ip traffic** command and returns a response.

Session ID	Action	Description
Telnet_myDUT.2		Telnet_myDUT.ffsp
Telnet_myDUT.2.1	open	telnet to 10.155.2.3:23
Telnet_myDUT.2.2	command	********
Telnet_myDUT.2.3	command	show version
Telnet_myDUT.2.4	command	show ip traffic
Telnet_myDUT.2.5	command	exit
Telnet_myDUT.2.6	close	telnet to 10.155.2.3:23

- When you start a session, iTest captures an **open** action (open the connection).
- When a session ends (for example, when you send an **exit** command in a Telnet session), iTest captures a **close** action (close the connection).
- The actions that *you* perform occur between the **open** and **close** actions. The session typically returns a response for each action.

Captured items

For each action, iTest captures the following items as a *captured item* (a single line on the Capture view): the action, the command, the response, and some additional information like timestamp and session identifier.

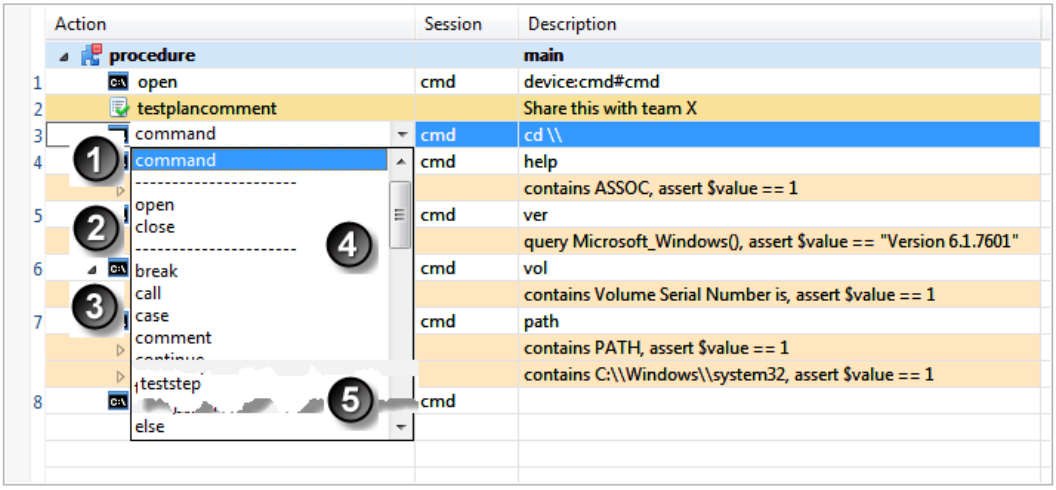
In the screenshot, the selected line represents one captured item in the **Telnet_myDUT.2** session. You could, for example, save the captured item as a single step in a test case.

Actions in test cases

While adding a step in the Test Case editor, you typically click in the **Action** cell to specify the action for the step (for example, to select the action that submits a CLI **command**, adds a **comment** step, or does an SNMP **get**).

Tip To view the description of any action, add a step and begin to type the action into the **Action** cell. As you type, iTest displays a list of actions. When you select an action, iTest displays a brief description with a link to more complete help.

When you add a step and open the list of actions, they are grouped as follows:



1 Actions that are specific to the session type

The actions in the first group depend upon the session type for the step.

Each session type has a particular list of appropriate actions. In the example, the **break** and **command** actions are appropriate for the Telnet session type.

Actions that are specific to a particular session type are described in the chapter for the session type.

2 Actions that open and close sessions

open (open the connection and start a session) and **close** (end a session and close the connection) are special actions that appear for all session types. The first step for any session is always an **open** action.

See "The open action: Start a session" on page 241 and "The close action: Close a session" on page 243

3 Executive (EXEC) actions

The actions in this group appear for all session types and are called EXEC (executive) actions. EXEC actions are general-purpose flow-control, looping, and administrative actions that enable you, for example, to:

- Control the order or flow of step execution (for example, **for** and **while** loops and **if** constructs)
- **call** procedures or **run** external test cases and **return** results from them

- **synchronize** execution threads
- Perform other special operations like add a **comment** or a message that should appear in test reports, **read** in the text of a file, **write** out to a file, **configure** a traffic generator, or **summarize** test results.

See "iTest EXEC actions, grouped by function" on page 244.

4 Custom, user-defined actions: QuickCalls

In addition to the built-in actions provided by iTest, you can create custom actions called QuickCalls. Your QuickCalls will appear here, between the **open**/**close** actions and the EXEC actions in the list. QuickCalls are very powerful because you can use them both during interactive testing and in automated test cases. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

5 Action step to export the action steps (test plan) to a text-based format

The action step **teststep** is part of the Exec actions, general action command and appears for all session types. This allows you to export the test action steps (your test plan) to a text file and if required, share them with a group/team within the organization.

See "The 'teststep' action: Export the test plan to a text file" on page 255.

Session-control actions: open and close

The open action: Start a session

An **open** step in a test case starts a session using the device or session profile that is specified in the **Description** cell. Each step in the session has the same **Session** (session ID) value (**DUT6_Telnet** in the example).

Action	Session	Description
procedure		**main**
open	DUT6_Telnet	project://project/session_profiles/DUT6_Telnet.ffsp

To add an open step

- #### Fastest and Easiest way: Save a captured session

  The fast and easy way to add an **open** step is to save a captured session into a test case. The fully-configured **open** step is added as the first step in the session. You do not need to do anything else.

- #### Add the open step manually

  Alternatively, when you add an **open** step by selecting **open** in the **Action** cell, you will specify additional values in the **Session** and **Description** cells, as described here:

1 Identify the session by its **Session** name:

When you add an **open** step, the **Session** name for the step is populated with the **Session name** property from the session profile (**DUT6_Telnet** in the example). If the session profile does not specify a value for the **Session name** property, then you must specify a unique name in the **Session** cell to identify the session that the **open** step starts. All steps in the session will use the same **Session** name. **Session** names can start with a letter, underscore, or number.

2 Specify the session to open. Click ... in the **Description** cell to open the **Select a Session Profile or Device** dialog box. Specify one of the following methods for opening a session:

- **Topology or testbed device:** The box displays each device associated with the current test case (either a global topology or testbed or the topology or testbed specified for the test case). Select the device from the list and then click **OK**.

  The text in the **Description** cell will be a device URI of the format **device:device_name**. For example, **device:telnet_DUT6** means: **"**From the global testbed or the testbed specified for the test case (on the Test Case editor **General** page), fetch a device definition named **telnet_DUT6**."

- **Session profile or reference session profile:** The box displays a tree of all projects in the current workspace. Navigate to the session profile and then click **OK**.

  The resulting text in the **Description** cell will be the URI of the session profile that will start the session, for example, **project://my_project/session_profiles/telnet_DUT6.ffsp**

  For instructions on configuring this value to be determined at runtime, see "Determining the device or session profile (dynamically) at runtime" on page 242.

- **iTest default session type**: This is a powerful option that you can use when generalizing a test case. You can use a parameter in the **Session** or **Description** cells for a step so that the session ID, device, or session profile can be determined at runtime. With this option, you must specify a **Default session** for the step or procedure. See "Determining the device or session profile (dynamically) at runtime" on page 242.

3 For additional settings that you can configure and information on using parameters in **open** steps, see "Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step" on page 183.

Determining the device or session profile (dynamically) at runtime

One powerful way to make a test case portable (that is, reusable over a variety of topologies or testbeds without any changes required) is to use parameters to supply the session profiles that the test case should use when opening sessions.

Parameterizing the session profile in an open step

1 Create a testbed document (**File > New > Other > iTest > Testbed**).

2 On the Testbed editor **Devices** page, add a session profile. On the Test Case Editor **Steps** page, add an **open** step. Now that you have specified a Local testbed, the device appears as a choice in the drop-down menu for the **Description**.

At this point, the test case has some flexibility. Whenever you specify a different testbed, you can select the appropriate device in the **open** step. You can also specify that the test

case should use the Global testbed so that the test case “picks up” whichever testbed is in use.

The close action: Close a session

The **close** action closes the session specified by the **Session** property value (and opened by an associated **open** action).

For **close** steps, the **Description** cell (the value of the **Command** property) is blank.

Typically, your test case should include one **close** step for each **open** step with the same **Session** value.

Actions for CLI session types

iTest supports several flavors of CLI (command line interface) session types — we will describe the CLI **command** action and **break** action in this section.

The command action: Submit a command

The **command** action is available only for CLI sessions and submits the text that appears in the **Description** cell (the value of the **Command** property). **command** is the most commonly used action in CLI test cases.

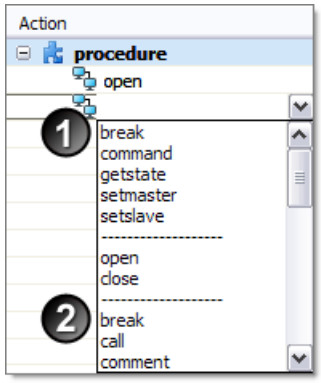
Note By default, iTest sends a carriage return + linefeed sequence when the **Command** cell is blank, so there is no need to include **[char \r\n]** in the **Command** or **Description** cell for blank commands.

In the example, step 2 submits the **show ip traffic** command to the **3750telnet** session.

	Action	Session	Description
	procedure		**main**
1	open	3750telnet	project://my_project/session_profiles/telnet_DUT3.ffsp
2	command	3750telnet	show ip traffic

The break action: Send the break character

There are two distinct kinds of b**reak** action:



1 **Break CLI session execution**: The **break** that appears in the first group of actions appears in the list only for CLI sessions. For example, in a Telnet session, a **break** action stops the currently executing command immediately and execution then continues with the next step. The command sends the break character specified in the **Description** cell. Because many devices use Ctrl-C as the break character, the default is **[char Ctrl-C]**.

2 **Break out of a for, forEach** (*Tcl only*)**, or while loop**: See "The 'break' action: Break out of a loop" on page 328.

iTest EXEC actions, grouped by function

General actions

Loops, switch, and if/then logic

The following actions are described in Chapter 16, "Controlling execution flow: Loops, If/Then, and Switch":

- **Loop control**

- **For and ForEach loops**

  The for action: Execute a group of steps in a loop See page 321.

  The foreach action: Execute a group of steps in a loop See page 324.

- **While loops**

  The while action: Repeat the steps in a 'while' loop See page 325.

- **Switch logic**

  The 'switch' action: Control execution flow based on the value of a variable or expression (Tcl) See page 334.

- **If / Then / Else logic**

  The 'if' action: Element of an if/then or if-elif-else construct See page 329.

  The 'then' action: Element of an if/then construct (Tcl) See page 331.

  The 'else' action: Element of an if/then or if-else-elif construct See page 332.

  The 'elseif'/'elif' action: Element of an if/then or if-elif-else construct See page 333.

Actions for procedures

The following actions are described in Chapter 10, "Procedures":

The 'call' action: Calling a procedure See page 233.

The 'return' action: Returning execution from the current procedure See page 234.

The 'write' action: Adding text into the response of a call step See page 235.

Tip You should know about a powerful alternative to procedures: The iTest QuickCall feature makes it easy to add custom actions to the built-in iTest actions (e.g., **getTable** in SNMP sessions). You can use QuickCalls both during interactive (manual) testing and as test case steps. QuickCall procedures are custom actions that ***you*** define for a session profile or device. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

Actions that read from and write to files

The 'readFile' action: Return the contents of a file See page 256.

The 'writeFile' action: Write information to a file See page 257.

Actions that perform Tcl operations

The 'scriptEval' action: Evaluate a Tcl command See page 259.

The 'scriptGet' action: Get the value of a variable (Tcl or a selected interpreter) See page 261.

The 'scriptSet' action: Set the value of a variable (Tcl or a selected interpreter) See page 262.

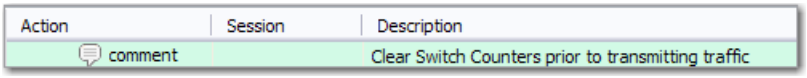
Asynchronous (threaded) execution

The following actions are described in Chapter 36, "Making your test case thread-safe: Actions that help you to synchronize execution threads":

General actions

The 'comment' action: Add a comment to a test case

For the **comment** action, iTest interprets the text in the associated **Description** field as a comment to the user. You have the option to specify that, at runtime, all comments in the test case should appear as execution messages in the Execution view and in the test report, but the step performs no other action.



Comment steps are a great way to outline or pseudo-code a test case before adding actual commands. To add comments quickly, select a **Comment** step and then press Ctrl-Enter. Once your outline is complete, you can go back and insert commands.

Wrapping associated steps under a comment step

To organize steps and keep a test case looking "clean", you can group associated steps under a **comment** step (the steps execute normally, they are merely indented under the **comment** step to improve readability). After grouping the steps, type appropriate text for the comment into the

Description cell. As a result, you can collapse or expand a group of steps under the parent **comment** step using ⊞ and ⊟. When you collapse all steps, the test case reads like pseudocode so that a coworker can easily understand the operation of the test case. To wrap steps, select them, right-click, and then select **Wrap in Comment**.

Using comments to “message out” to the Execution view

On the Test Case editor **General** page, select **Generate an execution issue for every comment step executed**. Comment text can include field replacements. This is an easy way to send data to the Execution view.

- **To enable substitution for a comment step**

In the **General** properties group for the step, check **For the Command field, perform command, variable, and backslash substitutions**.

To ensure access to certain data that is available when the message is generated, iTest first applies its standard field substitution and then uses Java-style format strings for messages. Java format strings uses escaping rules that differ from Tcl rules.

For example, Java string format uses single quote ' as its special character and you need two of these for escaping. So, to cause ‘ to appear in the message, use two single quotes ‘’ in the message text. For the Java string format rules, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/MessageFormat.html

The ‘message’ action: Add a severity type and message type

For the **message** action, iTest interprets the text in the associated **Description** field as a message with severity type to the user. At runtime, all messages in the test case appear as execution issue messages in the Execution view and test report.

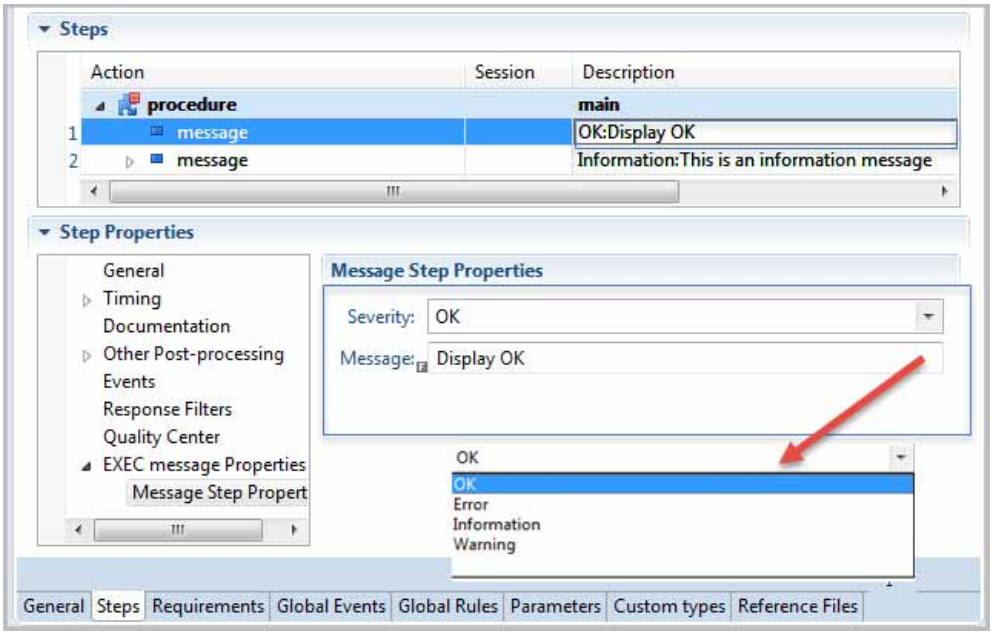
The message actions supports severity levels: **OK**, **Error**, **Information**, and **Warning**. You may either enter the severity type and message manually or select from **Step Properties > EXEC message Properties > Message Step Properties**page (see the screenshot “Action ‘message’” below).

- **Severity:** Drop-down list options:**OK**, **Error**, **Information** (default), and **Warning**
- **Message:** Enter the message to be displayed in the Execution view and test report.

Note The Message field on **Step Properties > EXEC message Properties > Message Step Properties** page supports field substitution.

The severity option and the message specified on the **Step Properties > EXEC message Properties > Message Step Properties** page are synchronized with the description contents and vice versa.

Action ‘message’



The ‘eval’ action: Evaluate an iTest interpreter command

An **eval** step evaluates the iTest statements specified in the **Description** cell.

Because **eval** operates on the whole command body (which can be multi-line), you can execute multiple statements with a single **eval** statement.

There is no **Session** associated with an **eval** Action.

Example:

This **eval** step sets the value of a local variable.

Tcl examaple:

Action	Session	Description
eval		set port 4

Python examaple:

eval		a = 1
eval		b = "5"
eval		type(a)
eval		type(b)

Note Using an **eval** action directs the statement to the iTest interpreter, which does not support all Tcl commands (and supports iTest commands that are not supported by Tcl). Typically, you should use an EXEC **eval** action. In cases where you need to use a Tcl command that does not exist in the iTest interpreter, however, use **scriptEval**. See “iTest interpreter commands in steps” on page 128.

About the iTest interpreter and the Tcl interpreters

- **Actions that operate in the iTest environment: eval, set, get**

  The iTest interpreter performs tasks that are useful in the iTest environment. We designed the syntax to be very much like Tcl so that the commands would be easier to understand. You can use iTest interpreter commands to perform a variety of tasks; some commands set a variable value (**set**), get a variable value (**get**), perform mathematical operations (**math.abs**), return information about iTest (**info**), or access the response to an earlier step (**response**).

  Some iTest interpreter commands have Tcl counterparts and some do not. For example, you can use **set i 0** (which uses the same syntax as Tcl) to assign the value **0** to the variable **i**. You can then use **i** as a local variable in your test case with the value **$i**.

  You can use the iTest **eval** action to evaluate the iTest commands (actually, statements) that are specified in the **Description** cell.

The iTest interpreter and the Tcl interpreter

- **Using special iTest actions to access the Tcl environment: scriptEval, scriptSet, and scriptGet**

  The **scriptEval** action uses the Tcl interpreter. If you perform **set i 0**, then variable **i** is set only in the Tcl environment — the iTest environment does not know about the variable.

  Note Even though the variable that we just set is named **i**, it is a variable in the Tcl environment—a completely different and independent variable from the variable **i** that we earlier set in the iTest environment.

  You can continue to use **scriptEval** to, for example, source your own Tcl script that makes use of **$i**. The benefit of having **scriptEval** is that it supports all Tcl operations.

  To pass variables back and forth between the iTest interpreter and the Tcl interpreter, use **scriptSet** and **scriptGet**.

  Note scriptEval, scriptSet, and scritGet are not application in Python

- **You can think of it as the “iTest world” and “Tcl world”:**
  - When you use the **scriptEval** action, you are operating in the Tcl world.
  - When you use the **eval** action, you are operating in the iTest world.
  - You use the **scriptSet** and **scriptGet** actions to peek and poke variables between the Tcl world and the iTest world.

About the iTest interpreter and the Python interpreters

Actions that operate in the iTest environment: eval

The iTest interpreter performs tasks that are useful in the iTest environment. The built-in Python interpreter starts with these modules: basic math functions, random number generation, time methods, regular expressions, file reads and writes, stdio, data structure manipulations, and JSON structure processing.

In addiiton to using the iTest interpreter commands to perform a variety of tasks, you may may assign global data structures (iTest QuickCall setting a global variable), which can be read and written to by Python interpreter (a test case in Python syntax that calls the QuickCall).

Some iTest interpreter commands have Tcl/Python counterparts and some do not. For example, you can use assign **i = 0** to assign the value **0** to the variable **i**.

You can use the iTest Python **eval** action to evaluate the iTest commands (actually, statements) that are specified in the **Description** cell.

The ‘run’ action: Execute the specified test case

A **run** step executes the specified test case (the ***child*** test case — sometimes called a foreign or an external test case) and optionally passes parameter values.

See “Executing a child test case: The ‘run’ action” on page 129.

Note If a **call** step in a child test case **B** (begun by a **run** step in a grandparent test case **A**) calls grandchild test case **C**: The **call**ed test case **C** will use the shared session from test case **A** in its **open** step if the **Session ID** in **C** is same as the **Session ID** in **A**. If you do not want to use the shared session, then change the **Session ID** in **C** to be different from the **Session ID** in **A**.

The ‘exit’ action: Stopping execution of the test case

An **exit** step immediately stops executing the current procedure and all threads to end test case execution.

The action does not change the existing execution result of the test case (Pass, Fail, Abort, or Indeterminate).

An appropriate execution message appears in the Execution view, in the Step Issues view, and in test reports.

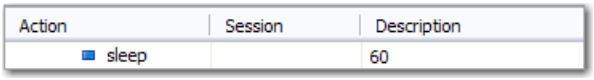
The exit action has no configurable properties.

The ‘sleep’ action: Pause execution of the current procedure

The **sleep** action pauses the current thread for the number of seconds specified in the **Command** field (default: 5 seconds).

Upon completion of the sleep period, iTest executes the next step in the procedure in step number order.

This example step implements a sleep interval of 60 seconds to allow enough time for restarting the device.



The ‘summarize’ action: Summarize the current test case execution results

The response to a **summarize** step lists the pass/fail result and identifying information on the test case and its child test cases and lists all execution messages for the run.The response to a **summarize** step is like the **Overview** section of a test report.

- The response to a **summarize** step appears in the Response view.
- The response is automatically mapped, so you do not have to create a response map.

Tip: Email the summary to your team

One popular use of the **summarize** action is to email the execution summary to your team.

a Add a **summarize** step. Save the response to the step into a variable using the **Store response in variable** property for the step.

b In a **Mail** session, use a **response** field replacement to write the summary into the body of an email message.

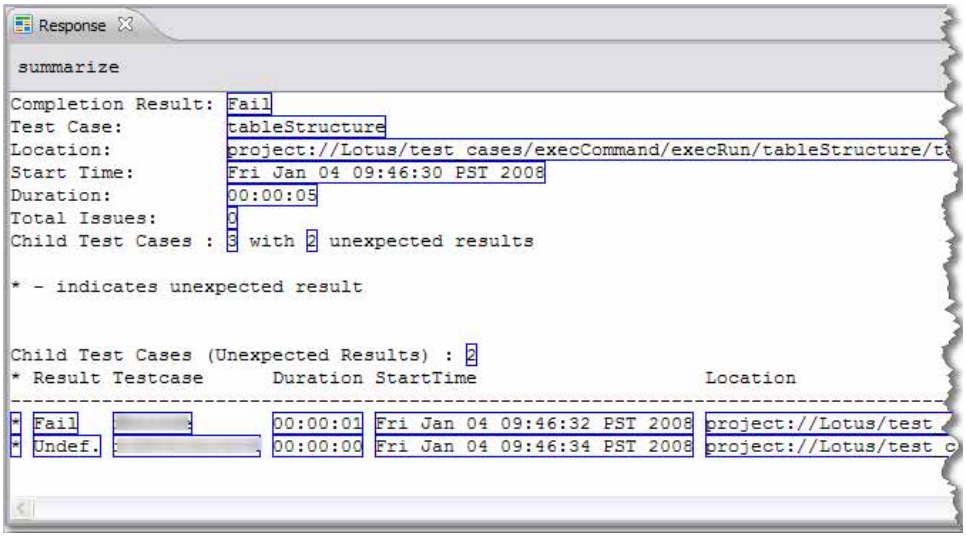
Summaries for procedures that execute child test cases

If the **summarize** step is in a procedure that executes child test cases, then the response to the **summarize** step includes a table of expected and unexpected results for the child test cases. **“Unexpected results”** means any value other than the value that you specified for the **Expected completion result** property for the test case (for example, a **Fail** result for a test case that you expect to have a **Pass** result).

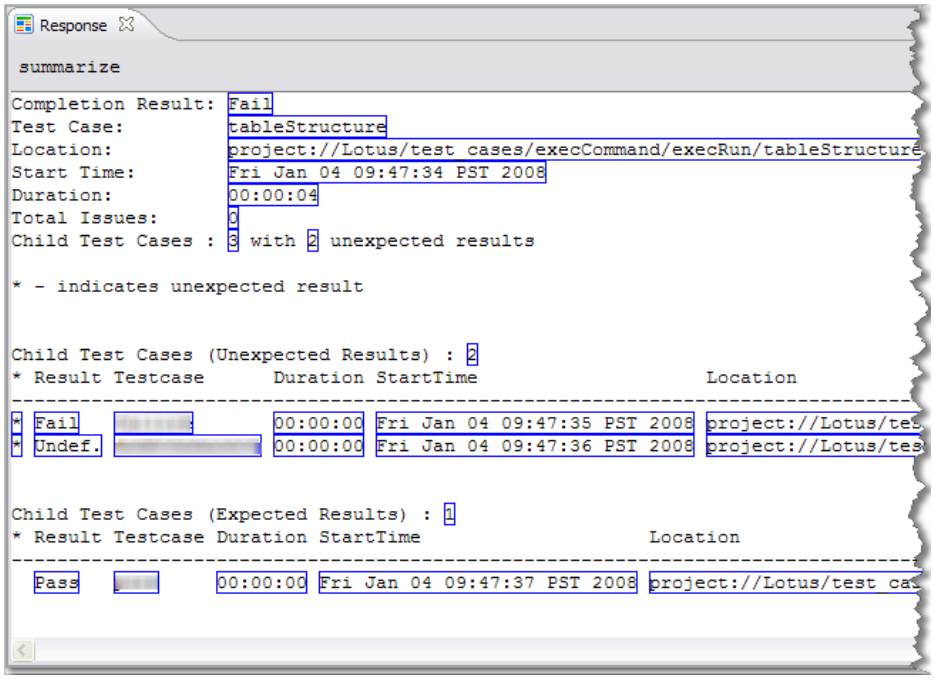
Format of the response to a summarize step

Instructions for specifying the format of the response appear in Step 3 of “To add a summarize step” on page 253. You can specify the following response formats:

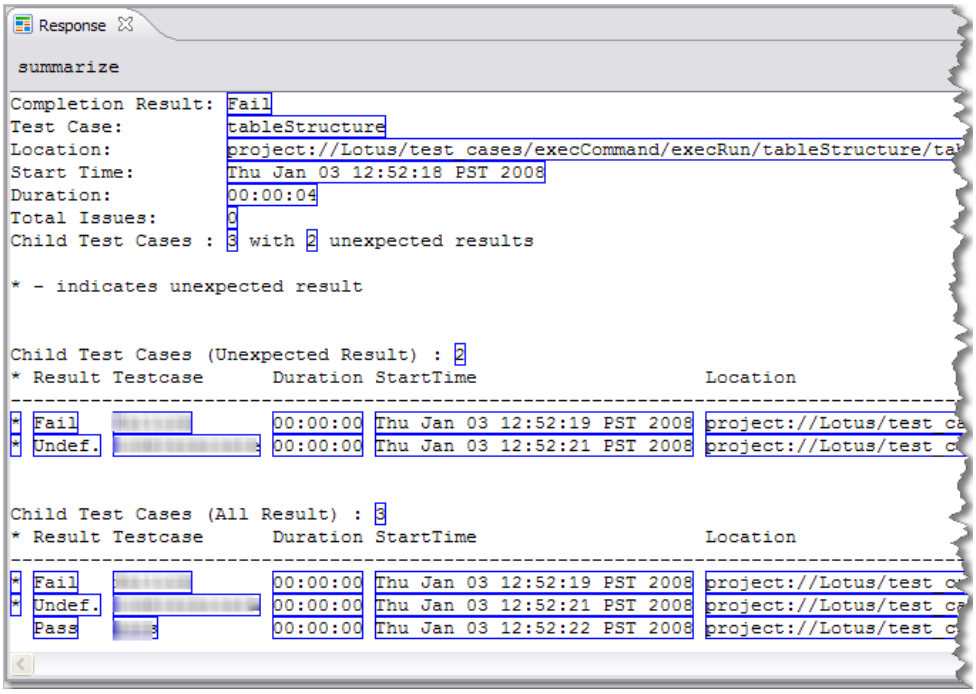
List only tests with unexpected results



Display two tables: unexpected followed by expected



Display two tables: unexpected followed by all tests (this is the default format)



- **To add a summarize step**

1. Add a step and select an **Action** of **summarize**.
2. Leave the **Description** cell (the value of the **Command** property) blank.
3. In the **Step Properties** section, open the **EXEC summarize Properties > Summarize** properties group and specify the **Response format** for the **summarize** step (as described above).

The ‘dumpState’ action: Return Threads view content, Data view content, and a summary of execution results

The response to a **dumpState** step (commonly used for troubleshooting) can include any or all of the following information:

- Execution thread information (the data that would currently be displayed in the Threads view)
- The data that would currently be displayed in the Data view
- The identical content as is returned by a **summarize** step

---

Note The response to a **dumpState** step appears in the Response view.
The response is automatically mapped, so you do not have to create a response map.

- **To add a dumpState step**

1. Add a step and select an **Action** of **dumpState**.
2. Leave the **Description** cell (the value of the **Command** property) blank.

3 In the **Step Properties** section, open the **EXEC dumpState Properties > dumpState step Properties** group and specify the following settings:

Append Threads view content	Check the box to add execution thread information (the data that would currently be displayed in the Threads view). The heading **Threads View Summary:** appears before the content. Default: checked
Append Data view content	Check the box to add the data that would currently be displayed in the Data view. The heading **Data View Summary:** appears before the content. Default: checked
Append summary	Check the box to add the identical content as is returned by a **summarize** step (as described in "The 'summarize' action: Summarize the current test case execution results" on page 251. The heading **Execution Summary:** appears before the content. The text appears in the default format for **summarize** steps: **Display two tables: unexpected followed by all tests**. Default: checked

◆ Tip: Email the 'dumpState' data to your team

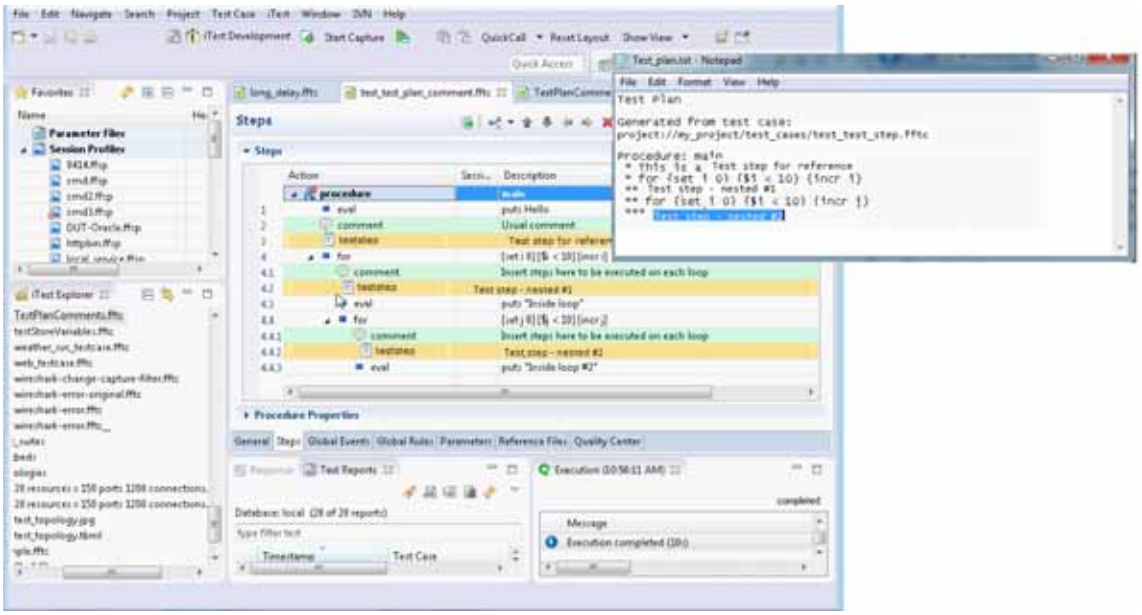
One popular use of the **dumpState** action is to email the execution summary to your team.

a Add a **dumpState** step. Save the response to the step into a variable using the **Store response in variable** property for the step.

b In a **Mail** session, use a **response** field replacement to write the data into the body of an email message.

The ‘teststep’ action: Export the test plan to a text file

If the action step **teststep** is included, you may create manual test plans in iTest and fill in the automation steps later. This assists with ease of automating test cases as the details for each automation step is included with iTest. In addition, you may export the test action steps (your test plan) to a text file and share the exported test plan with members of your organization.



You may include **teststep** as required to clarify the steps in the test plan being exported. Add and verify testplantcomment as below.

1. Create a new test case or select an existing test case
2. Add steps which includes **teststep**
3. Add loops or commands, as required.

   After adding all the required steps, click **Export test plan** icon.
4. Save the test plan to a desired location.
5. Navigate to the saved test plan location and verify that the teststeps and relevant steps (loops, etc) are saved in the text file.

Actions for loops and if/then logic

The following actions are described in Chapter 16, “Controlling execution flow: Loops, If/Then, and Switch”:

- **Loop control**

  The ‘break’ action: Break out of a loop See page 328.

  The ‘continue’ action: Interrupt a loop iteration See page 329.

- **For and ForEach loops**

  The for action: Execute a group of steps in a loop See page 321.

The foreach action: Execute a group of steps in a loop See page 324.

- **While loops**

The while action: Repeat the steps in a 'while' loop See page 325.

- **If / Then / Else logic**

The 'if' action: Element of an if/then or if-elif-else construct See page 329.

The 'then' action: Element of an if/then construct (Tcl) See page 331.

The 'else' action: Element of an if/then or if-else-elif construct See page 332.

The 'elseif'/'elif' action: Element of an if/then or if-elif-else construct See page 333.

Actions for procedures

The following actions are described in Chapter 10, "Procedures":

The 'call' action: Calling a procedure See page 233.

The 'return' action: Returning execution from the current procedure See page 234.

The 'write' action: Adding text into the response of a call step See page 235.

Actions that read from and write to files

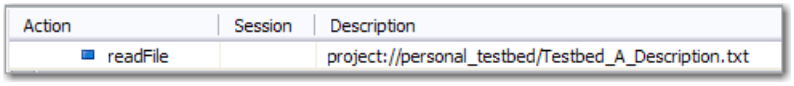
The 'readFile' action: Return the contents of a file See page 256.

The 'writeFile' action: Write information to a file See page 257.

The 'readFile' action: Return the contents of a file

An EXEC **readFile** step returns the text or binary data of the file whose path is specified in the **Description** cell (the value of the **Command** property).

For example, the response to the following step is the text of the **Testbed_A_Description.txt** file.



Tip For text files you can use **readFile** to obtain data and then use analysis rules for the step to save appropriate data items into variables for use later in the test case (using **get**, **${***varName***}**, **gget**, or **${/data/***varName***}**, as appropriate). In the text of the file, you might use delimiter characters (for example, commas or colons) to delimit data values to make extraction easier in the analysis rule.

Adding a readFile step

1 Create the step with an **Action** of **readFile**. Do not specify a session.

2 In the **Description** cell (or **Command** property) for the step, specify the file. To enable the use of typical URI and directory path syntax, field replacements are not supported in this field.

You can type or paste the URI or file path of the file, or, more typically, do either of the following:

- Click **Browse** ... in the **Description** cell
- In the **Step Properties** section > **General** page, Click **Browse** for the **Command** property

The **File Selection** dialog box opens. Specify one of the following:

- **Workspace** (the file is in the current iTest workspace)
- **File system** (the file is not in the workspace, but is somewhere on the file system)

Browse to the file and then click **OK**. iTest adds the URI of the file to the **Description** cell.

3 In the **Step Properties** section, open the **EXEC readFile Properties > ReadFile** properties group and specify settings for the following properties:

Content type	**Text:** A text file (you specify the encoding type using the **Encoding** property) **Text/xml:** An XML?format file **Text/html:** An HTML?format file **Text/tl1:** A file containing a valid TL1 message or sequence of messages. **Note** For more information on TL1 steps, see “Configuring sessions and test case steps for TL1 devices” on page 1215. **Binary:** A file containing arbitrary binary data. **Default:** Text
Encoding	Optional. Specify the encoding type so that the text file can be properly read or parsed. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. **Note** **Encoding** is ignored if **Content type** is set to **Binary**. Default: UTF-8

The ‘writeFile’ action: Write information to a file

Note The **writeFile** action is supported for TCL only.

You can use a **writeFile** step to write information to a file while executing a test case. For example, create a file that stores variable values, or data that is extracted from a response, or the entire text of the response to a step. Here are some options for **writeFile** actions:

- You can overwrite or append to an existing file
- You can add a blank line to the end of the text file (after the end of the data)
- For multiline text data, you can specify the delimiter character to use to separate lines (cr, cr/lf and so on)

- When the text information consists of multiple values, you can specify the delimiter character (comma, newline, and so on) to use to separate values

Adding a writeFile step

1. Create the step with an **Action** of **writeFile**. Do not specify a session.
2. In the **Description** cell, type or paste the URI or file path of the file to write to. Type a space character and then type or paste the contents to write to the file. Field replacements are supported. For example, the file contents to write can be **[query myMultivaluedResponse <myQuery>]** where **myMultivaluedResponse** is storing a response
3. In the **Step Properties** section, open the **EXEC writeFile Properties > writeFile** properties group and specify settings for the following properties:

Note The **writeFile** action is supported for TCL only.

If file exists	This property applies only if the specified file already exists at the time that the test case tries to write to it. Specify whether to overwrite (replace) the file or to append the specified data to the end of the file. Default: Append
File type	This property specifies how file writing is performed. • **Text**: The data to be written is treated as a sequence of lines. Depending on other property values, line delimiters can be changed on file writing. • **Binary**: The data from specified variable is written as is, without any processing Default: **Text**
Add a new line at the end of the file	Check the box to add a blank line to the end of the text file after writing the specified data into the file. iTest will add the characters specified in the **Line delimiter** property Default: checked **Note** If **File type** is set to **Binary** then this property is ignored
Line delimiter	Optional. This property applies only if the specified text file content consists of multiple lines. Specify the delimiter character to use to separate lines of data in the file. Default: Use the default delimiter for this platform **Note** If **File type** is set to **Binary** then this property is ignored
Encoding	Optional. Specify the encoding type for text data to use so that, later, it can be properly parsed. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8 **Note** If **File type** is set to **Binary** then this property is ignored

Actions that perform Tcl operations

About the iTest interpreter and the Tcl interpreters

- **Actions that operate in the iTest environment: eval, set, get**

  The iTest interpreter performs tasks that are useful in the iTest environment. We designed the syntax to be very much like Tcl so that the commands would be easier to understand. You can use iTest interpreter commands to perform a variety of tasks; some commands set a variable value (**set**), get a variable value (**get**), perform mathematical operations (**math.abs**), return information about iTest (**info**), or access the response to an earlier step (**response**).

  Some iTest interpreter commands have Tcl counterparts and some do not. For example, you can use **set i 0** (which uses the same syntax as Tcl) to assign the value **0** to the variable **i**. You can then use **i** as a local variable in your test case with the value **$i**.

  You can use the iTest **eval** action to evaluate the iTest commands (actually, statements) that are specified in the **Description** cell.

The iTest interpreter and the Tcl interpreter

- **Using special iTest actions to access the Tcl environment: scriptEval, scriptSet, and scriptGet**

  The **scriptEval** action uses the Tcl interpreter. If you perform **set i 0**, then variable **i** is set only in the Tcl environment — the iTest environment does not know about the variable.

  Note Even though the variable that we just set is named **i**, it is a variable in the Tcl environment—a completely different and independent variable from the variable **i** that we earlier set in the iTest environment.

  You can continue to use **scriptEval** to, for example, source your own Tcl script that makes use of **$i**. The benefit of having **scriptEval** is that it supports all Tcl operations.

  To pass variables back and forth between the iTest interpreter and the Tcl interpreter, use **scriptSet** and **scriptGet**.

  Note scriptEval, scriptSet, and scritGet are not application in Python

  ---

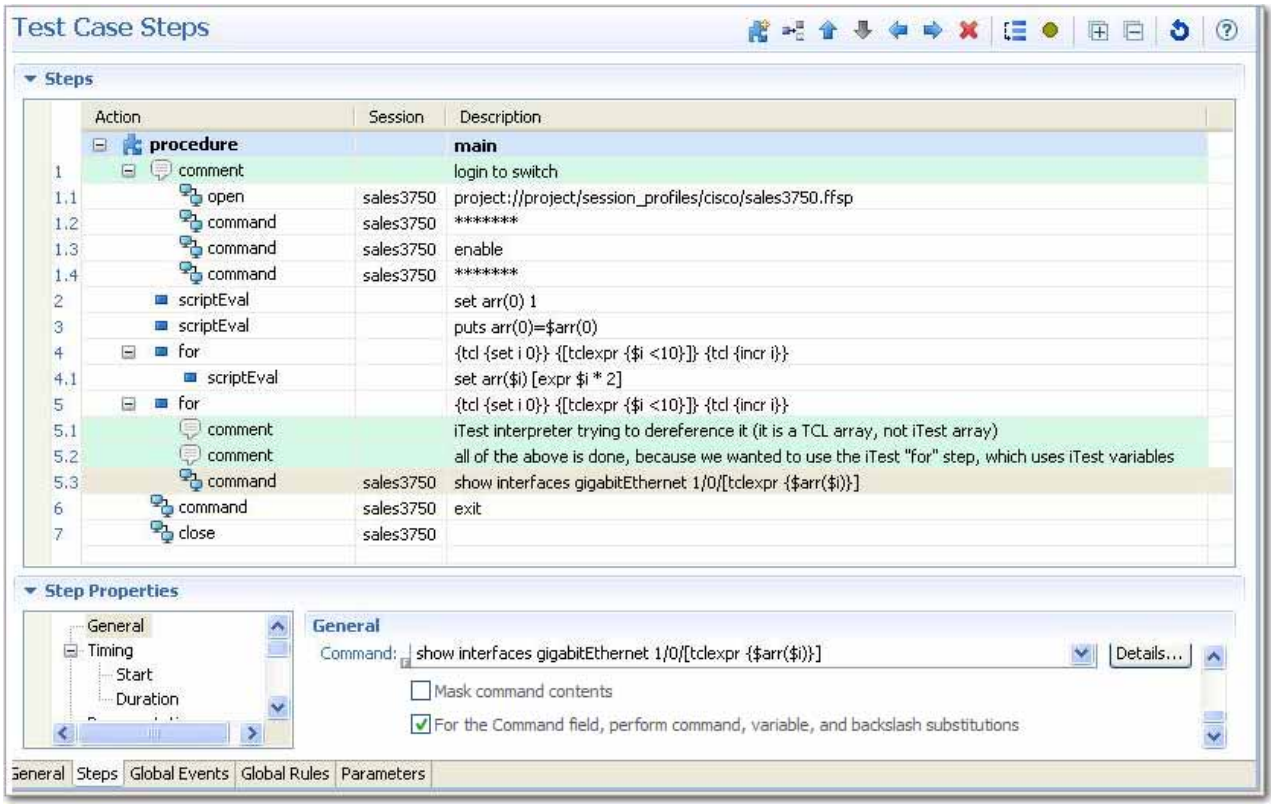
- **You can think of it as the “iTest world” and “Tcl world”:**
  - When you use the **scriptEval** action, you are operating in the Tcl world.
  - When you use the **eval** action, you are operating in the iTest world.
  - You use the **scriptSet** and **scriptGet** actions to peek and poke variables between the Tcl world and the iTest world.

The ‘scriptEval’ action: Evaluate a Tcl command

The **scriptEval** action evaluates the Tcl command specified in the **Description** cell (the value of the **Command** property) using the Global Tcl interpreter.

The Tcl command can be multi-line (a script).

Example



Substitutions in the Command text

By default, the following types of substitution are not made to the text of the **Command** property before the step is executed:

- Command field replacements
- Variables
- Backslash characters used to escape special characters

To cause iTest to pre-process the **Command** text and perform such substitutions before the text is interpreted as a Tcl statement, follow this procedure:

1 Select the step.

2 On the **Step Properties** page, check the **For the Command field, first perform command, variable, and backslash substitutions** box.

Response

The response is populated (including structured data) in a way consistent with how a Tcl Shell Command step works (including result, STDOUT, and/or STDERR).

Moving array variables from the Tcl shell into iTest

To move an array variable **tcl_variable** in the Tcl shell into the iTest variable **itest_variable** to determine the value of **tcl_variable,** use the following steps to return an array list:

```
scriptEval array set tcl_variable {apple orange mango banana}

scriptGet itest_variable {[array get tcl_variable]}

puts $itest_variable
```

To access the array elements using **eval** in iTest, use:

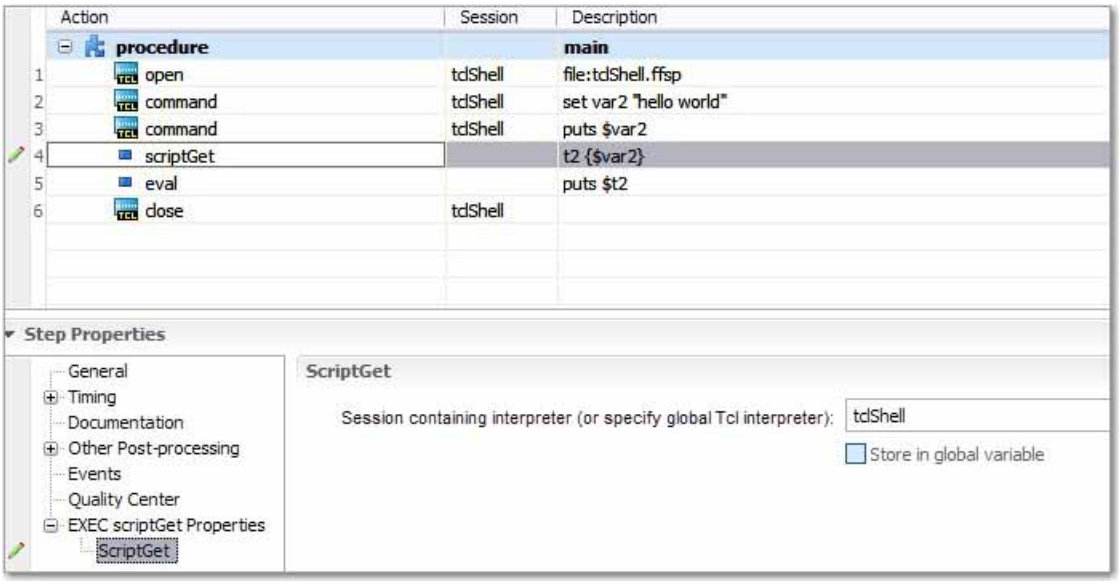
```
scriptGet itest_variable {[array get tcl_variable]}
eval array set me $itest_variable
puts $me(apple)
```

The ‘scriptGet’ action: Get the value of a variable (Tcl or a selected interpreter)

scriptGet gets the value of a variable from the specified interpreter and sets the specified iTest interpreter variable to the value. (By default, the command gets the value from the Global Tcl interpreter, but you have the option to specify the session with the target interpreter.)

scriptGet takes two arguments: the name of an iTest variable to be set; and something that is substituted by the Tcl interpreter. Command substitution happens on both arguments *before* the Tcl interpreter is asked to interpret the second argument.

In this example, **t2** is the iTest variable to get the value, and **var2** is the Tcl variable whose value will populate **t2**. The braces around **$var2** prevent substitution, causing it to be passed to the Tcl interpreter as the string “**$var2**“. Notice that we set the Session containing Tcl interpreter property for the step to the session for which we are doing the **get**.



Note While **scriptGet** does support getting individual array elements, you cannot use it to move array values between the Tcl interpreter and the iTest interpreter. Instead, use **scriptEval**.

Response

The response to a **scriptGet** step is the string representation of the variable value.

Adding a scriptGet step

1. Create the step and select an **Action** of **scriptGet**. (Do not specify a **Session ID**.)
2. **scriptGet** takes two arguments in the **Description** cell. The contents are interpreted as a list, and each element in the list is substituted (unless surrounded by curly braces).
3. Optional: Follow this procedure to obtain the value from a selected interpreter other than the Global Tcl interpreter (<itest_var> <expression>):

itest_variable	Name of the iTest variable to be set (actually any heap query suitable for “set”). If the iTest variable existed before the **scriptGet** step, then its previous value is lost. This works for all variable types including lists and arrays.
expression	Expression that will be evaluated by the specified Tcl interpreter. The iTest variable's value will be set to the result. To cause the expression to be treated as a string to be passed to the Tcl interpreter, surround it with { } braces.

In the **Step Properties** section, open the **EXEC scriptGet properties > scriptGet** properties group and specify settings as described here:

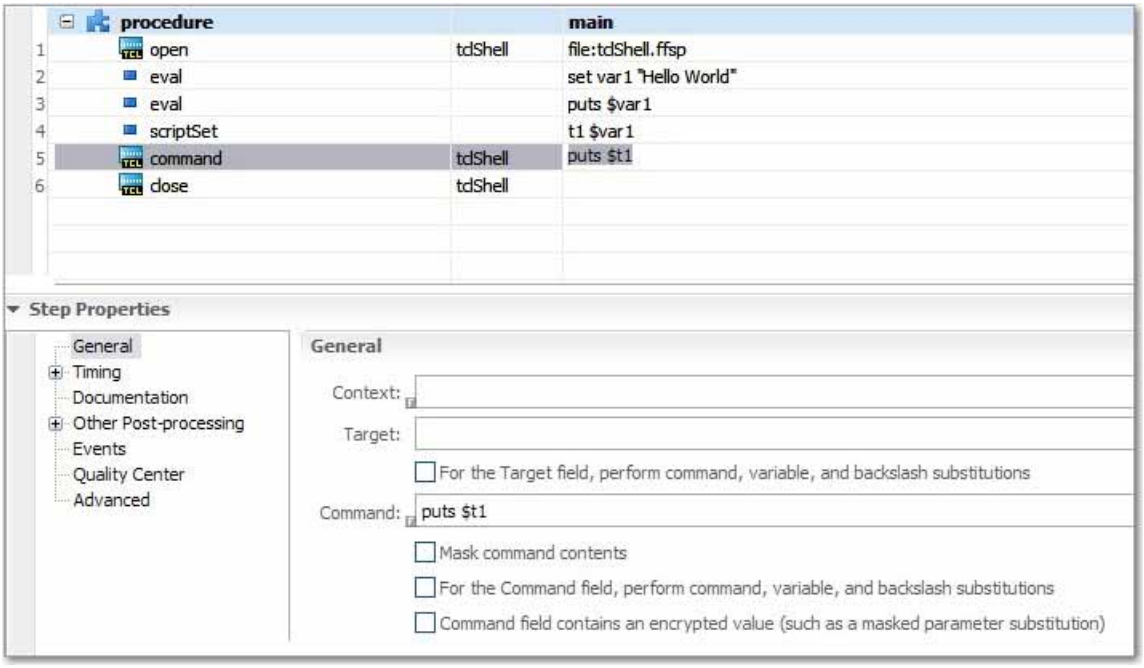
Session containing interpreter (or specify global Tcl interpreter)	Select the Session ID whose interpreter should be used for the **scriptGet** step. (If you select **Global Tcl interpreter**, then **scriptSet** obtains the value from the Global Tcl interpreter.) Default: Global Tcl interpreter
Store in Global variable	Check the box to store the variable in a global rather than local location.

The ‘scriptSet’ action: Set the value of a variable (Tcl or a selected interpreter)

The **scriptSet** action sets the value of a specified variable in the specified interpreter (typically, the Global Tcl interpreter, but you have the option to specify the interpreter). **scriptSet** works for any kind of variable, including lists, arrays, lists of lists, numbers, and so on. You can specify multiple values.

For example, an analysis rule has extracted a list of values. We have set the value of a iTest interpreter variable called **var1** with the list. To pass the list of values into a Tcl interpreter variable called **t1** in the Global Tcl interpreter's context, you can use the following **scriptSet**

step. Notice that we disabled substitution for the step by unchecking the **For the Command field, perform command, variable, and backslash substitutions** property.



Note While **scriptSet** does support getting individual array elements, you cannot use it to move array values between the Tcl interpreter and the iTest interpreter. Instead, use **scriptEval**.

Response

The response to a **scriptSet** step is the string representation of the variable value.

Adding a scriptSet step

1 Create the step and select an **Action** of **scriptSet**. (Do not specify a **Session ID**.)

The command for **scriptSet** steps is directed at the iTest interpreter. To ensure that iTest commands (like [tcl] or [tclexpr]) will be correctly interpreted, the property that controls field replacements (command substitution) for the step is disabled and dimmed (The **For the Command field, perform command, variable, and backslash substitution** checkbox is unchecked).

2 **scriptSet** takes two arguments in the **Description** cell (<itest_var> <value>):

itest_variable	Name of a Tcl variable in the specified interpreter whose value is to be set.
value	Value to set.

1 By default, **scriptSet** sets the variable value in the Global Tcl interpreter. Follow this procedure to set the value in the interpreter that you specify.

2 In the **Step Properties** section, open the **EXEC scriptSet properties > scriptSet** properties group.

3 Select the interpreter in the **Session containing interpreter** list. (If you select the default setting of **Global Tcl interpreter**, then **scriptSet** sets the value in the Global Tcl interpreter.)

Traffic generator actions

The ‘configure’ action: Configure a traffic generator device

Traffic Generator equipment sessions only

If you use traffic generators in your testbed, you probably spend considerable time configuring them in preparation for test execution. Once the device is correctly configured, you naturally save the configuration so you can quickly replicate it when needed.

To enable a step in a iTest test case to configure a traffic generator to the desired state before performing the automated test, iTest submits the same saved configuration information to the device to configure it. Here's how it works:

1. During an interactive session with the device, you use a **configuration get** step to obtain the traffic generator device's configuration settings. The response to the **configuration get** is the text of the configuration file.
2. Now, when you save a captured **configuration get** step into a test case step, iTest converts the **configuration get** step into a **configure** step. iTest places the text of the configuration file into the **Description** cell (actually, the value of the **Command** property) for the **configure** step.
3. Then, when the test case executes, the **configure** step submits the configuration settings, thus configuring the device exactly as if you had configured it using its native management console.

The **configure** action is typically the first step after **open** in Ixia Traffic and Agilent N2X sessions.

Creating a “Configure the traffic generator” step in a test case

1. Using the traffic generator device's native interface, configure it in the usual way.
2. Now, during an interactive iTest session with the device, submit a **configuration get** command. The configuration setting text is returned as the response to the **configuration get** command.
3. Save the captured **configuration get** step as a step in the test case. You can save any number of captured steps or the whole captured session, as long as the selection includes the **configuration get** step.
4. iTest converts the **configuration get** step into a **configure** step.

Making quick changes to the TrafficGen configuration settings

You can edit traffic streams or other settings in the TrafficGen configuration script without having to set up the traffic generator again. Follow these steps:

1. Back up the test case so you can recover if the changes do not work as you expect.

2 Copy the content of the TrafficGen.Configure step's Command cell into a text editor. In this example, we'll change the `vlanId` setting from `20` to **`90`**.

Search for the text -vlan:

```
"vlan                config       -vlanID                20"
```

Change the setting as needed, for example:

```
"vlan                config       -vlanID                90"
```

Actions for synchronous (threaded) execution

Synchronous (threaded) execution

The following actions are described in Chapter 36, “Making your test case thread-safe: Actions that help you to synchronize execution threads”.

lock: Ensure one thread for a specified block of code See page 801 in Chapter 36.

signalWait: Sleep the currently executing thread See page 802 in Chapter 36.

signalWaitAll: Wait until all specified events have been signaled or activated See page 803 in Chapter 36.

signal: Wake a thread that is waiting on an event See page 804 in Chapter 36.

signalAll: Wake all threads that are waiting on an event See page 804 in Chapter 36.

signalActivate: Turn a signal on See page 805 in Chapter 36.

signalClear: Deactivate a signal See page 805 in Chapter 36.

waitThread: Wait for steps to complete See page 806 in Chapter 36.

Also, see the related action: **The ‘dumpState’ action: Return Threads view content, Data view content, and a summary of execution results** See page 253.

CHAPTER 12

Session Windows

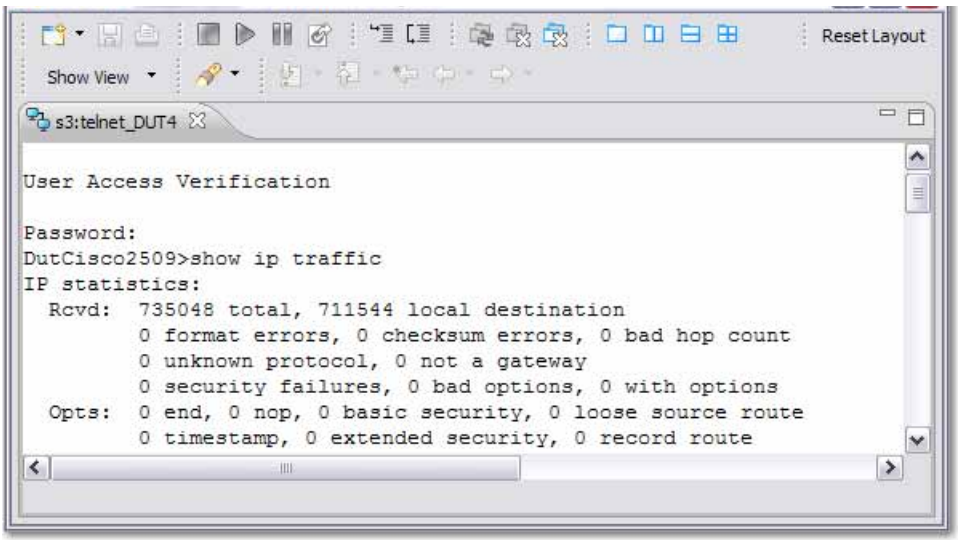
Session windows

iTest uses session windows to display executing test cases and your interaction with sessions on devices; terminal sessions for CLI session types (like Telnet, SSH, and a Tcl shell command interpreter for Tcl Shell sessions, for example), the appropriate type of browser for Web or Swing sessions, or a special MIB browser for SNMP sessions.

iTest supports the session types described in “iTest Overview” on page 1.

Session window for a CLI session type

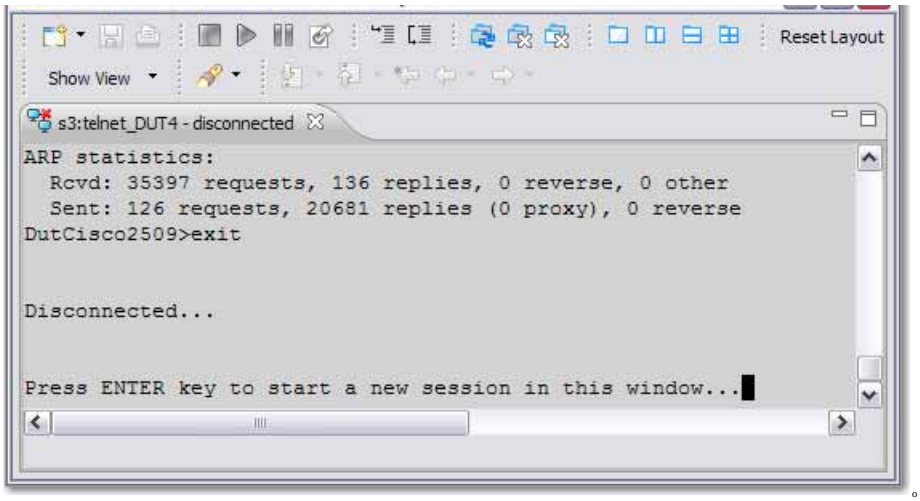
For example, the Telnet session window looks just like a Telnet client terminal session. When you type a command and press Enter, the editor displays both the command text that you submitted and the device’s response. (The command/response pair (plus some session information) is called a ***captured item*** and is listed in the Capture view.)



Each session window’s tab displays an icon that represents the session type, the unique **Session ID** (**s3** in the example), and the name of the session profile that was used to start the session (**telnet_DUT4** in the example).

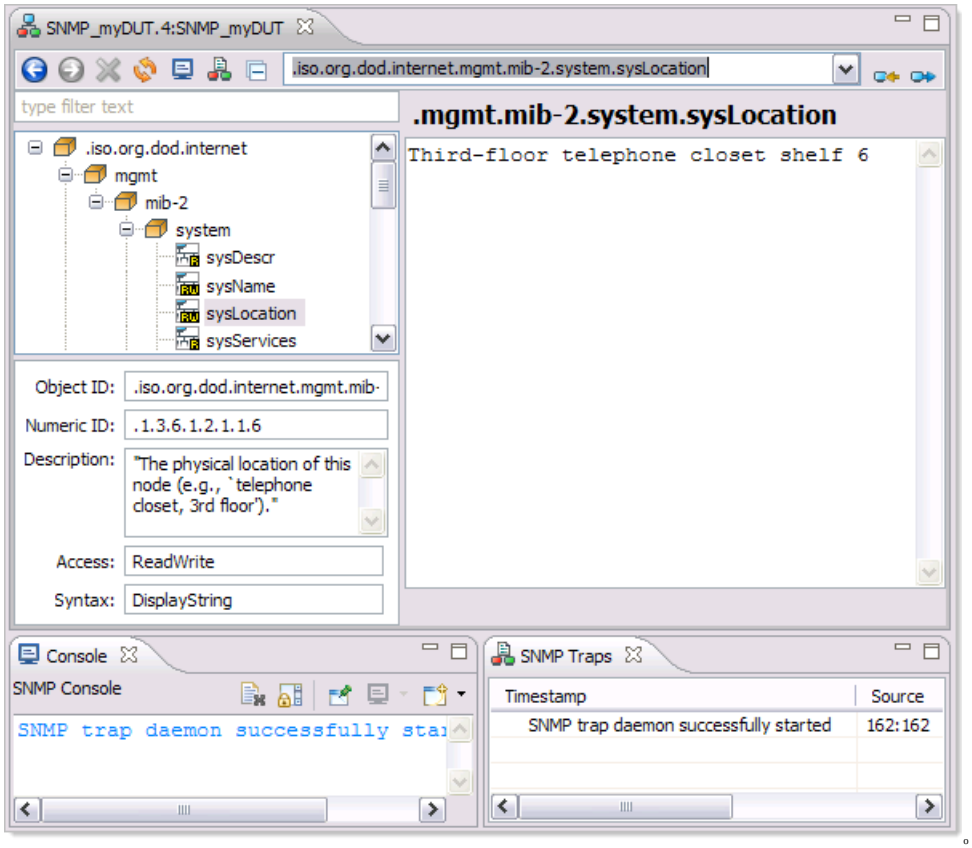
This example Telnet session with a device shows the login process, a **show ip traffic** command, and the device’s response — all just like a typical terminal display.

When you close a session (for example, by issuing an **exit** command), iTest captures a **close** Action and then dims the session window. At that time, you can save the session or steps from the captured session into a test case.



Session window for a browser-based session type

Session types like SNMP, Web, and Swing use the browser type that is appropriate for the application. For example, here is the browser that you will use when working in SNMP sessions:



Starting a session using a session profile

You can start an interactive session in any of the following ways:

- Double-click the session profile in the Favorites view
- Right-click the session profile in the he iTest Explorer and select **Start**
- Click **Start** in the Session Profile editor (the **New Session** page).

The session starts in a new session window, as described in “Session windows” on page 267.

Advanced Users Many of the property settings for session profiles support field replacements to enable you to parameterize settings so they can be determined dynamically at runtime. You might use **tcl**, **param**, or **profile** command field replacements to improve the flexibility and portability of automated test cases. Sometimes, to perform an interactive test, you might need to manually start a session that typically starts only for automated test sessions. To enable you to do this, if any **tcl**, **param**, or **profile** command field replacements are encountered while starting the session, iTest starts a Tcl interpreter so that the field replacement can be resolved. When the session ends, the Tcl interpreter is disposed. If a Tcl interpreter service is requested on restart, a new interpreter will be created and returned. See “Defining a session profile (configuring the session settings)” on page 73.

Starting CLI sessions on Linux and Unix

Because Linux and Unix devices typically include installed Telnet and SSH servers, you can start a Telnet or SSH session to **localhost** to access a shell.

Working with session windows

Rearranging session windows

When multiple sessions are active, it's sometimes hard to keep track of the various session windows. The following buttons in the main toolbar help you to manage session windows:

- To arrange the editors by splitting the editor area, click  or  or 
- To arrange the editors by stacking them on top of each other so that you can access a session by clicking its tab, click 
- You can drag session window windows between tab groups so that you can see multiple sessions at the same time.
- To give you even more room, you can start a new iTest window and start sessions in the new window – the sessions are all captured consistently, regardless of where the session window resides

Managing session windows

The following buttons in the main toolbar help you to manage session windows:

- To close all inactive session windows in the current window, click 

- To close all session windows (active or inactive) in the current window, click 

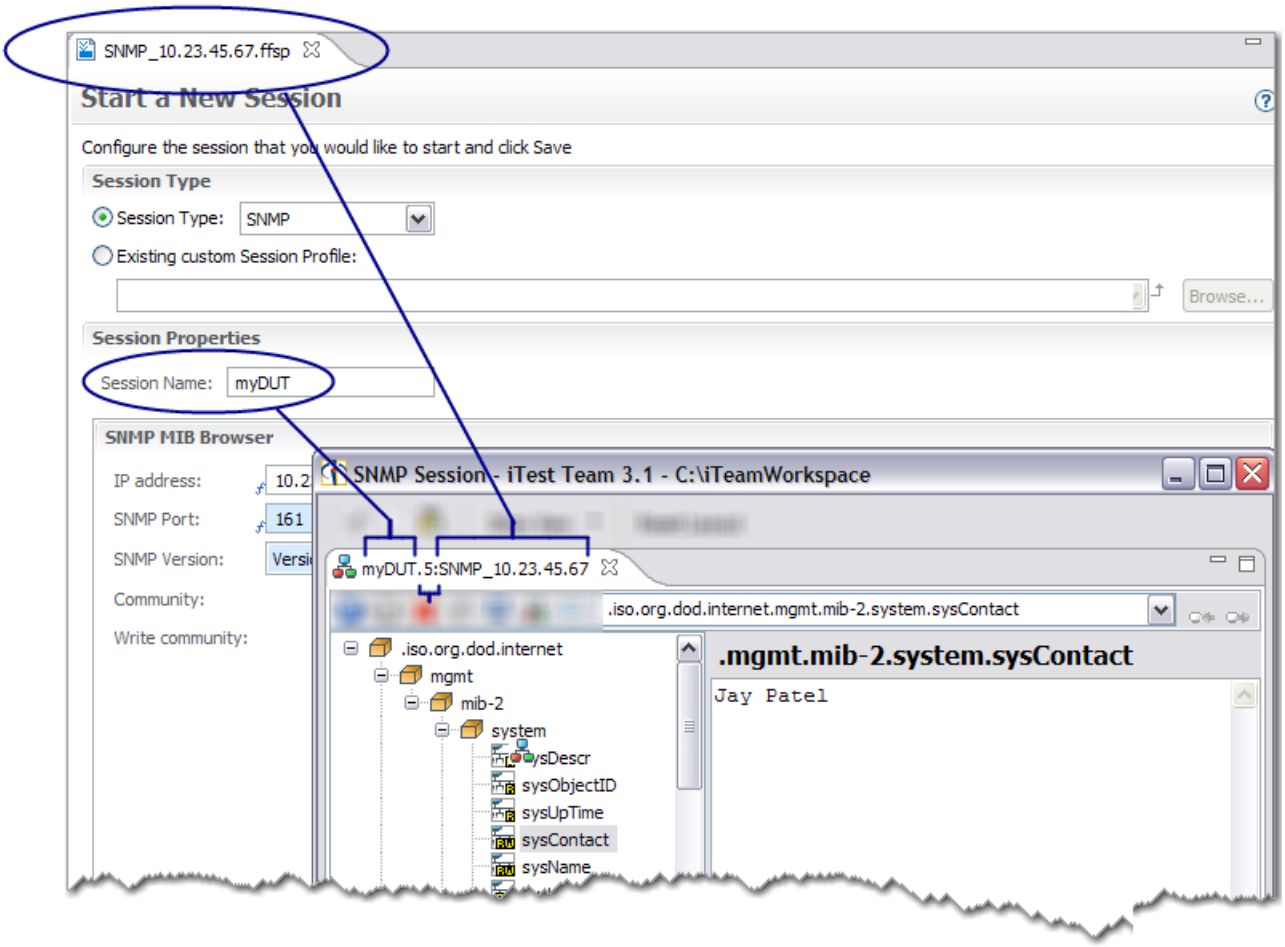
Inactive session windows

When a session is disconnected (that is, a **close** Action has occurred), the session window's icon changes to indicate inactive status and the window dims (becomes gray).

How sessions are named

The tabs for active session windows display a name that is generated from several sources: First, an icon reflecting the session type, then the **Session ID**, followed by a colon and the name of the session profile that was used to launch the session. In the example,

- The session profile is named **SNMP_10.23.45.67**
- The Session ID is **myDUT.5**



For devices with more than one session attached or for multiple captures that use the same session profile: To create the Session IDs that appear in the **Session** cells in the Test Case editor and in test reports, Spirent iTest uses the combination of **Session name** from the session profile (for example, **myDUT**) and a unique session number for the day. (for example, **myDUT.1** and **myDUT.2**). If the session profile does not specify a **Session name**, then Spirent iTest uses the filename of the session profile in its place.

CHAPTER 13

Executing Tests

Execution: Quick instructions

Whenever **Start Execution in New Window** is enabled, you can click it to execute the currently selected item.

To specify whether new sessions should start in the iTest window or in an existing session window, set the Execution preferences (see “Setting preferences for execution” on page 281). You can set other execution preferences as well.

Executing test cases

Use one of the following methods:

- In iTest Explorer, right-click the test case and select **Start Execution** , **Start Execution in New Window** , or **Load for Execution**
- In iTest Explorer, select the test case and then click in the main toolbar
- When the Test Case editor is active, click . You may be asked to save the test case before execution.

Scheduling execution

You can schedule tests to execute at a specified time in the future or to execute on a recurring basis once per day at a specified time on specified days of the week. See Chapter 18, “Scheduling Execution”.

Replaying a Capture report

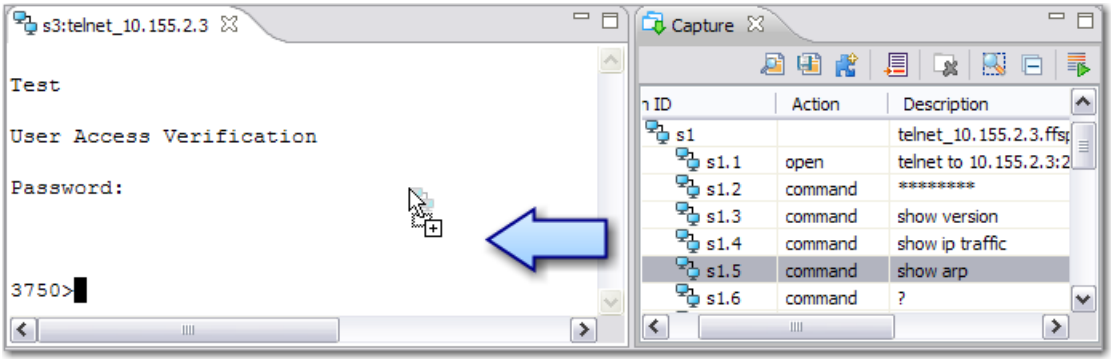
Use one of the following methods:

- In the iTest Explorer, right-click a Capture report and select **Start Execution** , **Start Execution in New Window** , or **Load for Execution**
- At any time while viewing a Capture report in the Capture Report editor, click

Replaying captured items from the Capture view

In the Capture view: If the group of captured items is all in the same session and *does not* include an **open** step, then drop the items into an active session window of the appropriate

session type. The items execute immediately. (You cannot include session **open** steps in the group of captured items.)



- A simple and powerful way to manually execute a series of CLI commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and then paste them into an active session. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.
- Alternatively, if a session window of the appropriate type is open, then select the items and click .
- If the selected group of captured items include an open item for each session, then click .

◆ **For a single selected item:**

- If the session from which the item was captured is still active, then the item is immediately executed in that session.
- If iTest determines that the item could be executed in any of multiple active sessions, then a dialog box asks you to specify the session in which to execute.

  Shift-drop the item into an active session of the appropriate session type. iTest pastes the command into the session window, but does not execute it. This option enables you to edit the command before submitting it.

Replaying captured sessions from the Capture view

In the Capture view, select a session or a group of sessions. Click .

Tip Because iTest captures commands and responses from captured items that you replay, it captures the commands and responses that are being replayed too. This means that you can create a new (more complex) Capture report or captured session by using the capture that results from executing multiple original Capture reports, captured sessions, and captured items.

Scheduling execution

You can schedule tests to execute at a specified time in the future or to execute on a recurring basis once per day at a specified time on specified days of the week. See Chapter 18, “Scheduling Execution”.

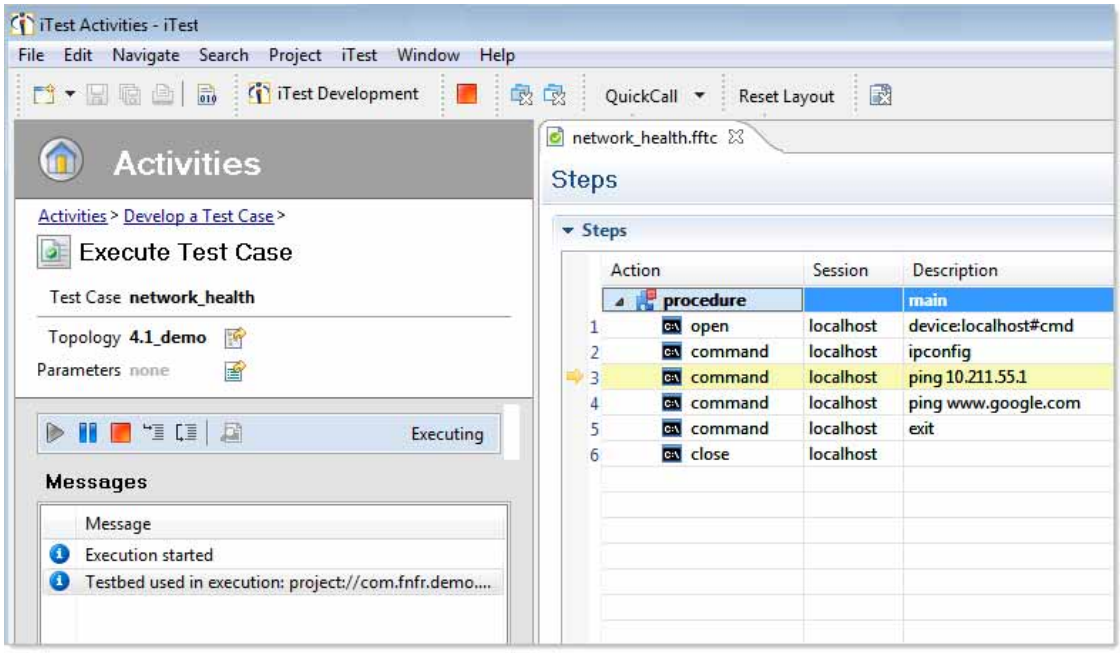
Loading a test case for execution

To make it easy to execute a test case one step at a time or one procedure at a time, you can load it for execution. iTest loads the test case, but does not begin execution. This makes it easy for you to single-step through a test case for debugging.

Once the test case is loaded, you can use the Data view to change runtime parameter values.

Keeping track of the currently executing step

During execution and when you pause execution, the Test Case editor highlights the currently executing step and indicates it with a yellow arrow in the left column. You can turn this feature on and off with a preference setting in **Spirent > General > Execution**.



Loading for execution

1 Load the test case using one of the following methods:

- Before starting execution, in either the **Execution** perspective or the **Test Case Debugging** perspective, click **Load for execution** .
- While working in the Test Case editor, save the test case and then click **Load for execution** .
- In the iTest Explorer, right-click the test case and select **Load for execution** or **Load for execution – New Window**.

2 The Test Case editor opens to the first step in the test case. Execution is paused.

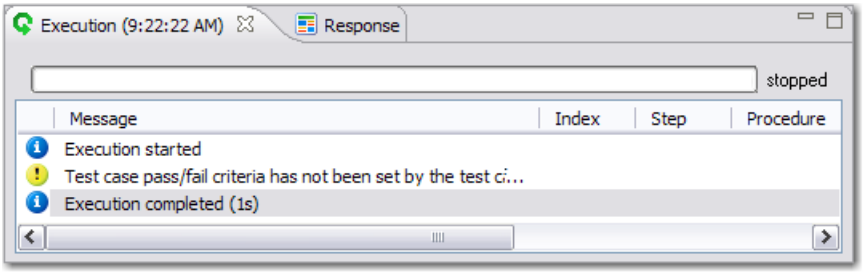
3 You can now change runtime parameter values in the Data view.

4 Use **Execute One Step** or **Execute Procedure and Pause** as needed.

Execution view

The Execution view opens when **execution issues** are generated during execution. Each issue has both a **severity** and a textual **execution message**.

- Icons represent the severity of the execution issues: **Information**, **Warning**, **OK**, and **Error**. Severity icons also appear in test reports and in the Step Issues view.
- Execution messages appear in the **Message** cell and describe execution events like test case pass/fail, results of analysis rules, execution pause/resume, and so on. Some execution messages are built-in (identifying the topology used for execution or generated by events — for example, “Execution started”); others are defined by the test case developer (generated by analysis rule actions — for example, “Only 3 ports of 4 are responding”). Execution messages also appear in the Step Issues view.
- For each execution issue, the view displays:
  - The step number for which the issue was generated
  - The source of the issue.
  - The step number of the step associated with the issue (from the first column in the Test Case editor's **Step** page).
- The **time remaining** bar displays the best estimate of the time remaining until the test finishes. The estimate is based on previous execution times and takes into account the optional execution speed setting. By default, tests run at the fastest possible speed.



Tip You can copy one or multiple issues to the clipboard. Select (Ctrl+click or Shift+click for multi-select) and then press Ctrl+C (or right-click and select **Copy**)

Icons indicate the severity of the issue	Each issue has a **Severity**: Information OK Warning Error
Message	The text of the execution message associated with the executed step. Some messages are built-in (for example, “Execution started”). Others are defined by the test case developer (for example, “Only 3 ports of 4 are responding”).

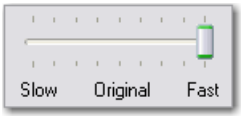
Index	The index number of the step associated with the message. **Note** **Index** and **Step** will often be the same. If a step is executed more than once during test execution, however, (for example, in a loop), then the index number increments by 1 for each iteration of the loop (2.1.1, 2.1.2...) while the step number remains the same.
Step	The step number of the step associated with the message (from the first column in the Test Case editor's **Step** page). **Note** **Step** and **Index** will often be the same. If a step is executed more than once during test execution, however, (for example, in a loop), then the index number increments by 1 for each iteration of the loop (2.1.1, 2.1.2...) while the step number remains the same.
Procedure	The name of the procedure in which the step executes.
Session	Session ID of the session in which the step executes.
Source	The iTest process that presented the message.
File	The file name of the file that contains the executed step. This may be the originally executed test case or a foreign (external) test case that was executed by a **run** step.

Execution view operation while working in the Test Report Comparison editor

While you are working in the **Test Report Comparison** editor, the Execution view displays the list of differences marked by the diff icon . Double-click a difference to select the corresponding difference on the **Comparison** page of the **Test Report Comparison** editor.

Execution speed control

By default, the speed control does not appear in the Execution view. To display the control, set the preference setting as described in the **Spirent > Execution** section of the **Preferences** page. See “Setting preferences for the Execution view” on page 276.



You can change the execution speed setting during execution to any one of the following settings or any setting in between.

Fast: Submit commands as quickly as the computer can send them while allowing each step run to completion before starting the next step. asynchronous (concurrent) steps, in contrast, are executed at original speed.

- **Original**: Execute no faster than the original capture. Execution may be slightly slower because each step will start no sooner than the original delay from the previous step end, but may be later because it will wait for the previous step to complete before starting.
- **Slow**: Use this setting to give you a chance to observe behavior that you might miss at a higher speed.

Setting preferences for the Execution view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Views > Execution View.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Views > Execution View

When executing, open the Execution view	Check the box to display help at the top of the Capture view. Default: checked
Display the execution speed slider in the Execution view	Check the box to display the speed control in the Execution view. Default: unchecked

Threads view

The Threads view enables you to monitor detailed thread execution progress. The Threads view is updated every few seconds even when the program is not paused. You can perform any of the following tasks:

- Review each thread's stack while execution is paused. New threads are collapsed by default.
- When execution finishes, you can scroll through the view to review execution.
- Use the Threads view to monitor steps that execute concurrently (asynchronously, also commonly referred to as asynchronous execution, async execution, concurrent execution, or concurrency).

Thread synchronization information

To support test case development and debugging, the following columns in the Threads view display synchronization information:

Awaited signals	Displays the names of any signals that must be activated for the step to proceed. The signal can come from a **signal**, **signalAll**, or **signalActivate** step. For **signalWaitAll** steps, all of the signals in the list must be active before the step can proceed.
Owned locks	Displays the names of all locks that the step owns — locks that were set by the step or by a parent step.
Awaited lock	Displays the name of the lock that must be released for the step to proceed.

For more information on synchronizing threads, see Chapter 36, “Making your test case thread-safe: Actions that help you to synchronize execution threads”.

Data view

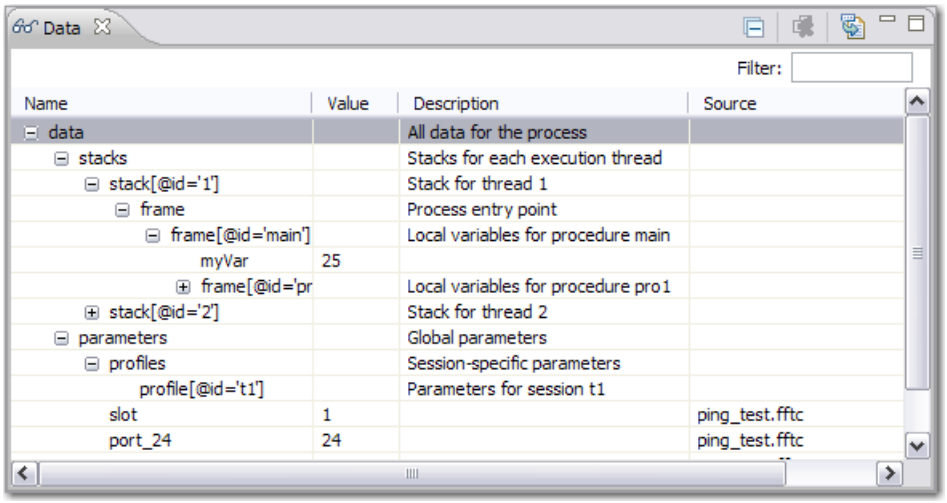
The Data view is used when developing and troubleshooting iTest test cases. When an execution is underway and paused in current workbench window, the Data view shows all or a portion of the data model being used by that execution. A toolbar button can be used to toggle through different subsets of the data.

All data in iTest is stored in a single hierarchy. This includes global variables, local variables, procedure arguments, parameters, and even copies of the topology or testbed and session properties.

When debugging a test case, you can view all of this data using this view. And, in fact, you can even modify the value of data items while execution is paused. For example, if you want to temporarily modify the value of a parameter while execution is underway, you can do that in this view.

The Data view offers two modes (click  to switch between the **Parameters** mode and the **Stack** mode.)

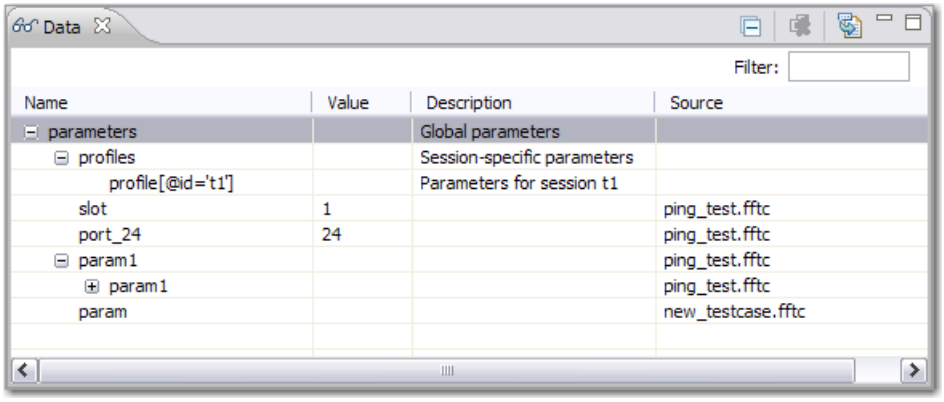
Parameters mode



In the **Parameters** mode, you can:

- View all parameters and both global and local execution variables (as they appear in the heap)
- View where they came from (in the **Source** column)
- View current local variables for each thread (listed under the thread name).
- Change parameter and variable values before executing a test case and while execution is paused. To change a parameter setting, select the value in the view and type a new value. (The **Description** text for each parameter is taken from the **Description** property, as specified on the **Parameters** page.)
- View session profile parameter values for each session (listed under the session name).

Stack mode



In the **Stack** mode, you can view variables for a stack frame in each thread.

Editing parameters and variables before and during execution

You can edit items in the Data view in either of the following cases:

- The test case is loaded for execution (select the test case and then click)
- While execution is paused.

The Data view displays variables under the **data** node in the heap (instead of the **stack** section). This is what makes the variable global. In contrast, local variables are created in the **stack** node in the heap. The **stack** section is transient, that is, it can be "popped" off and therefore lose all variable information.

How the Data view works:

During execution, the following actions occur:

- When the test case to be executed is first loaded, the parameters defined in the test case are merged into the execution heap.
- When a foreign test case is loaded (typically on a **run** step that refers to a test case file that has not yet been loaded), then the parameters from the foreign test case are merged into the execution heap.
- On an **open** step, the parameters associated with the session profile are integrated into the heap at **/parameters/profiles/profile[@session='{session}']** where **{session}** is the ID of the session about to be opened (after perhaps being overridden because of parameter merging, or because you explicitly changed one of the properties). On a **close** step, the node is deleted.
- During execution, the Data view shows the heap as it existed before execution was started/resumed. When execution pauses, the contents of the view are updated and editing is enabled. There is no restriction on which elements' values can be edited, however you cannot add new elements or delete existing elements.

In many cases, you may want to keep changes that you have made to the parameters so that the same values will be used in the next execution. For this reason, iTest retains the parameters section of the heap between executions, as long as the same test case is being executed in the same workbench window. See the descriptions for the **Reuse parameters between executions** and **Clear All Data** buttons.

Data view toolbar

Collapse All	Collapse the indented items into the top-level headings.
Clear All Data	Clear the heap. Discard all parameters so that the next execution runs as if for the first time.
Reuse parameters between executions	By default, each execution starts with cleared heap. To reuse parameters between executions, click **Reuse Parameters**. This setting enables you to keep changes you have made to the heap during debugging for the subsequent run. This setting is reset when you restart iTest.
Toggle Stack/Parameters	Switch between the parameters view and the stack view.

Replay

To ***replay*** a captured command is to have iTest execute a captured item from the Capture view as if you had typed the text or clicked the mouse or pressed the Enter key while working in a session. (You can replay whole captured sessions in the same way.)

Note We use the term **replay** only when discussing the execution of the captured session or a captured item from the Capture view. Otherwise, we'll use the term **execute**.

During replay, the following activities occur:

- Each affected session window's background color changes to indicate that replay is happening.
- The session windows display the commands and responses as they happen.
- The Capture view displays the captured items as they happen (iTest always captures).
- The Progress view displays messages if appropriate.
- The Execution view displays execution progress, the estimated time until completion of execution, and any applicable execution messages.

Replaying a Capture report

Use one of the following methods:

In the iTest Explorer or Favorites view, right-click a Capture report and select **Start Execution** or **Start Execution In New Window**.

At any time while viewing a Capture report in the Capture Report editor, click .

Replaying captured sessions

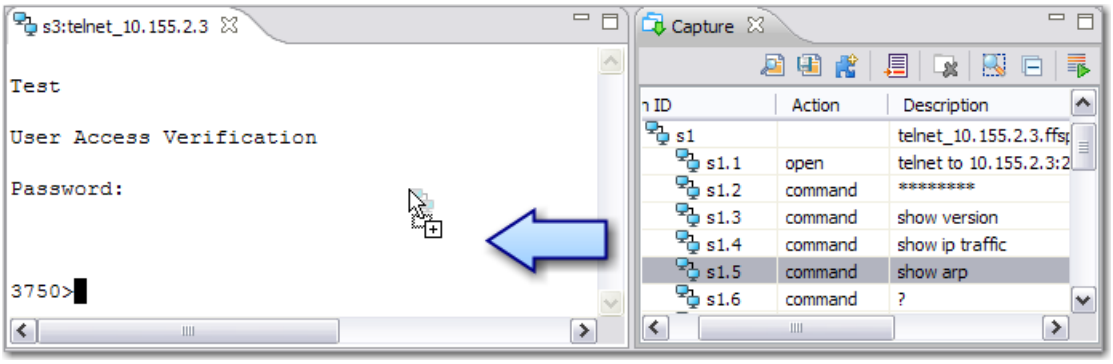
Select one or more captured sessions the Capture view.

In the Capture view toolbar, click .

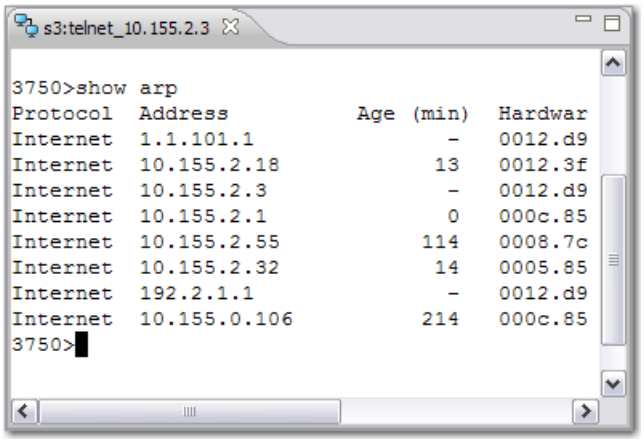
The sessions replay immediately.

Replaying captured items by dropping

From the Capture view, drop selected captured items into an active session window of the appropriate session type. The items execute immediately.



The command executes as if you had typed it.



Editing steps before replay

Shift-drop a single captured item into an active session window of the appropriate session type to paste the command into the session window, but not to execute it. This option enables you to edit the command before submitting it.

Tips

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and Paste them into an active session. The commands execute immediately.

- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Setting preferences for execution

Tip If you plan to run a large number of test cases for a long period of time, you can avoid out-of-memory errors (jave.lang.OutOfMemoryError: unable to create new native thread) by setting the **-Xss** JVM argument in the **iTest.ini** file. For example, **-Xss256K** or **-Xss128K**. The lower number allows more threads to be created.

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > General > Execution**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > General > Execution

Do not open session windows during execution	To improve performance, do not open the session windows that display the flow of execution. Default: unchecked
Before executing, close all inactive session windows	Before starting execution, end all active sessions and close all session windows. Default: unchecked
When execution finishes, close all Session windows started by the execution	If checked, then, once execution finishes, end all sessions and close all session windows. Default: checked
When execution finishes, automatically open Test report	If checked, then, display the Test Report in the place of the session window when execution ends. Default: checked
Bring Console Log view to front on execution start	Display the Console when a test case executes. Default: checked
Clear Console Log view contents on execution start	Clear all content from the Console before starting execution. Default: checked
Console buffer size (bytes)	Specify how much memory to allocate to the Console view. When the data reaches the limit, the oldest data is deleted. Default: 100,000
Default delay (seconds) for asynchronous steps set to High speed	Specify the default delay in seconds for steps that are both: asynchronous (the **Start this step (in a new thread) and proceed to the next step** property on the **General** properties page is selected) and Set to **High** execution speed Default: 2
Do not warn if saving editor during execution	Check the box so that iTest saves all changes to a test case while executing it without displaying the “Save the test case before executing” confirmation dialog. Default: unchecked
Do not warn if documents need saving before execution	Check the box so that iTest saves all changes to a test case before executing it without displaying the “Save the test case before executing” confirmation dialog. Default: unchecked
Action when the Local topology differs from the specified Global topology	Specify your preference on which testbed to use whenever the Local topology or testbed (the topology or testbed specified on the Test Case editor **General** page) is not the same as the Global topology or testbed. **Ask for topology choice**: Display a dialog box that asks the user to specify which topology to use. **Use Global topology**: By default, use the Global topology. **Use Local topology**: By default, use the Local topology. Default: Ask for topology choice

After execution, update "Estimated execution time" property in test case	The Estimated execution time property on the General page of the Test Case editor provides the best estimate of duration of execution for the test case. The value is used in two ways: • On the Execution view during execution, to determine the progress on the Time remaining progress bar • For use by test case scheduling applications Check the box to cause iTest to update the estimate when execution ends. The algorithm is (0.7*<current value>) + (0.3*<duration of latest execution>). The Estimated execution time value is not modified if: • User cancels execution • Test result is Failed or Abort • Execution speed setting is not "fastest possible" Default: checked
Include all steps in test reports (ignore the setting for the step)	Specifies that all steps should appear in test reports, even though individual steps might be configured not to appear (the **Include this step and its children in test reports** property is unchecked for the step). See "Controlling which executed steps appear in test reports" on page 439. Default: unchecked

Spirent > General > Execution > Tcl Interpreter

These settings specify the global Tcl interpreter used by the iTest kernel during execution

Interpreter	**Note** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: 1. If an interpreter is specified in the **Use the specified Tcl interpreter** property, then use that interpreter. 2. Otherwise, launch the first installed Tcl interpreter that iTest finds in the PATH environment variable. 3. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Built-in**: Use iTest's internal JACL Java-based Tcl interpreter. Because JACL does not support any C/C++ extensions, most traffic generator devices will not work in this interpreter. **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: **Auto-select**
Log Tcl commands to a console	Check the box to cause iTest to log the submitted commands to a console window. **Note** This setting is used only if you have specified a Tcl interpreter other than the built-in interpreter. Default: unchecked

Log Tcl responses to a console	Check the box to cause iTest to log the responses to Tcl commands to a console window. **Note** This setting is used only if you have specified a Tcl interpreter other than the built-in interpreter. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

CHAPTER 14

Reserving devices for execution

Overview

A reservation provider accepts a request to reserve all resources (devices and applications) in a topology to ensure that only one test at a time can use the resources. iTest assumes that reservations are **Active**; the resources have been allocated to the requester and are available for use.

Note iTest does not include a reservation provider software. You may obtain Spirent **Velocity**, which you may use to configure resources (devices and ports) and topologies (configurations of resources), reserve resources and topologies for testing/execution. Contact Spirent Customer Support for further information.

Making a reservation (Reserving a resource or topology for execution)

Making a reservation means:

1. You (the requester) request a reservation: You specify (to the reservation system) the requirements for resources and the timeframe that the resources are needed.
2. Depending on availability of the resources at the specified time, the system either:
   - Accepts the request and adds it to the reservation queue as **pending**

     or
   - Rejects the reservation request

Note The resources identified in a reservation request are not actually made available to the requester until the reservation is *activated* (as described in a moment).

To make a reservation

1. Do one of the following:
   - With the topology open in the **Topology** editor, right-click on the canvas and select **Reservations > Reserve**
   - With the topology open in the **Topology** editor, click **Topology > Reserve**

2 On the **New Reservation** dialog box, provide the following information and then click **OK**. All of the information will appear in the Reservations view for this reservation.

Name	Specify a meaningful name for the reservation
Duration	Specify the time period during which the topology should not be available for tests
Topology	Select the topology to reserve.

3 A confirmation message appears. If the reservation could not be made, the message provides information that should help you to change the request so that the reservation can be made.

Releasing a reservation

Typically, a test case releases a reservation when execution ends.

You also have the option to release a reservation when it is no longer needed.

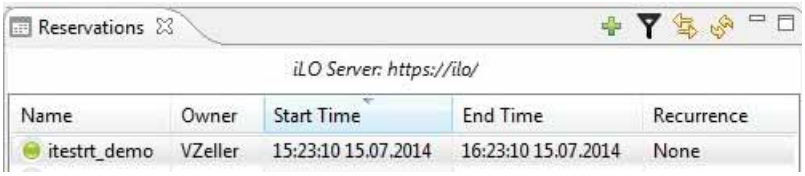
- **To release a reservation**

  With the topology open in the Topology editor, right-click on the canvas and select **Reservation > Release**. (Alternatively, in the main menu, click **Reservation > Release**.)

Reservations view

Right-click on the canvas and select **Reservations > View** to open the Reservations view.

The view lists all reservations from all reservation providers.



Double-click an item to open the topology in the Topology editor.

CHAPTER 15

iTest Views

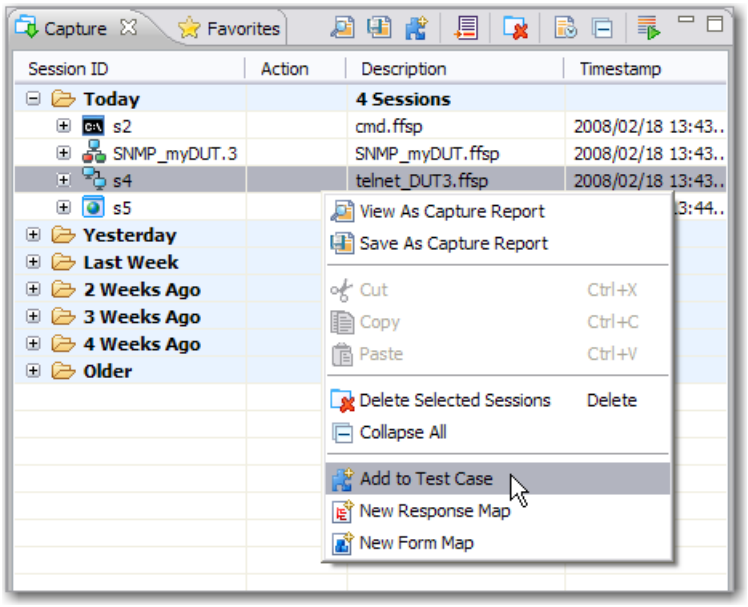
Views

Views are tabbed panes on the iTest window. Views provide information about iTest files and provide information that supports you while you work in the editors. For example, the Capture view displays information about the actions in a session that are currently being captured and that have been captured in earlier sessions.

Note The topics that describe most views appear in the appropriate chapter. For example, the chapter on SNMP sessions includes a section that describes the SNMP Traps view. This chapter describes the more general views.

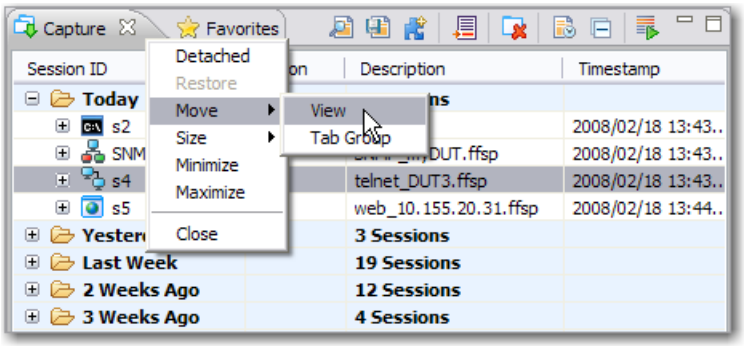
A view might appear by itself or be stacked with other views in a tabbed notebook.

- Click a tab to activate a view.
- Most views have toolbars and context (right-click) menus for actions on selected items.
- Press **F1** for online help for the current view.



- Double-click the tab to maximize a view or editor. Double-click it again to minimize it.

- Right-click a view's tab to access a menu that enables you to manage the view

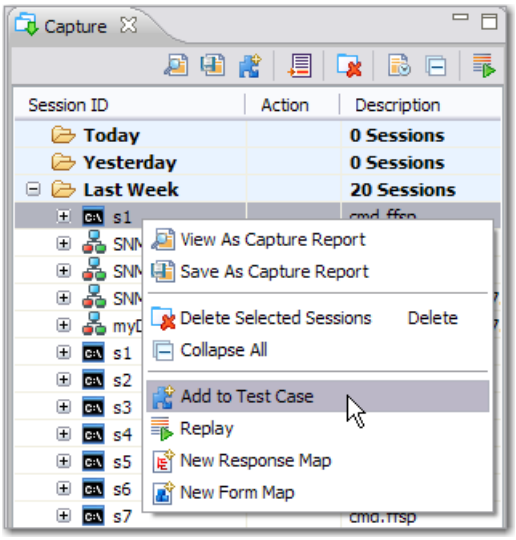


Tip To detach a view for placement anywhere on the screen, right-click the view's tab and select **Detached**.

When in doubt, right-click

Most of iTest's editors and views include context (right-click) menus for common operations (most toolbar tools appear in the context menus).

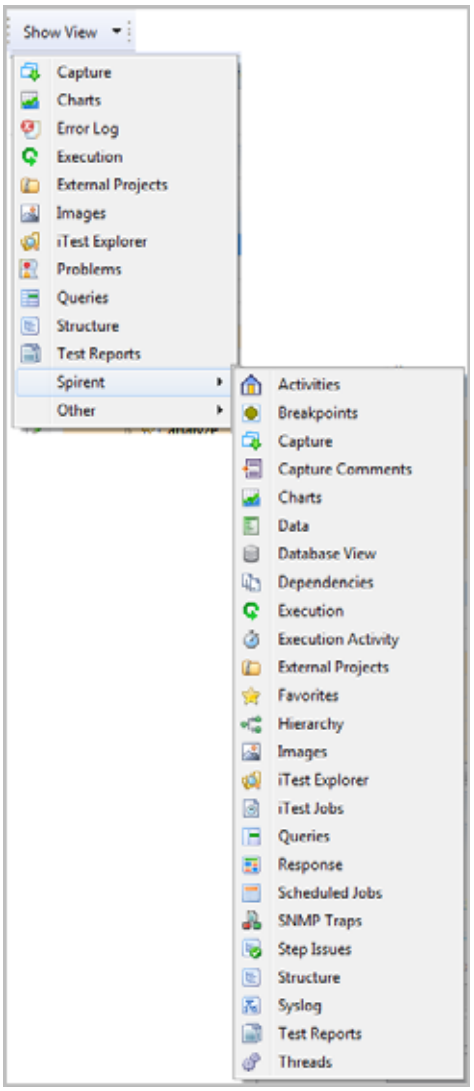
- For example, here's the context menu in the Capture view:



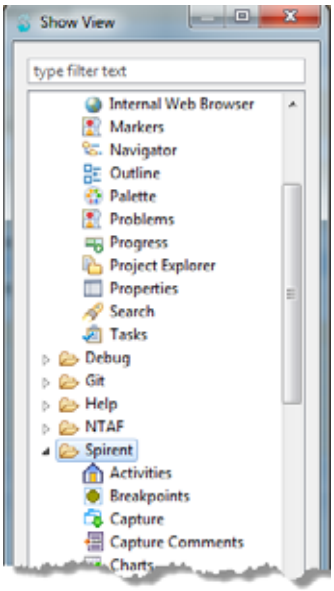
Opening a view

Use one of the following methods to open a view:

- In the main toolbar, click the arrow on Show View and select the view from the drop-down list. If the view is not listed, then select **Spirent** and then select the view.



- To select from a list of all available views, click the arrow on Show View. Select the view in the **Show View** dialog box.

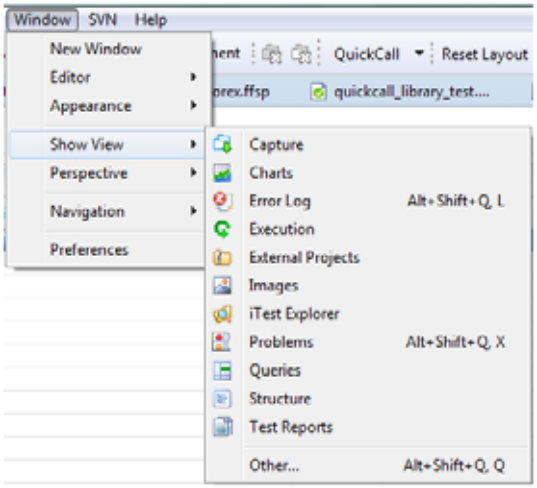


- Select it from the **Window > Show View** menu.

Displaying a view

Use one of the following methods to display a view:

- In the main toolbar, click Show View and select the view from the drop-down list. If the view is not listed, then select **Spirent** and select the view.
- To select from a list of all available views, click Show View. On the **Show View** dialog box, select the view.
- Select the view from the **Window > Show View** menu.



Working with perspectives

When you perform particular tasks, iTest will present a dialog box asking you whether it's OK to switch to another perspective. It's a good idea to switch, because the recommended perspective is optimized for the task that you're starting.

(If you decide to switch back to the perspective you started with, select it from the list in the upper right of the iTest window, as described in "Switching between perspectives" on page 53.)

Tips

While switching to the other perspective, if you checked **Remember my choice** not to be asked again, you can reverse that decision by setting a preference to view the dialog whenever a perspective is about to switch. See "Setting preferences for perspectives" on page 54.
In addition, you can set other preferences for perspectives on the same page.

Response view

The **Response** view displays the command and response for the currently selected captured item or test case step. For test case steps, the text is the response received for the step, when it (step) ran in the most recent execution of the test case. (You can select captured items in the Capture view or Capture Report editor. You can select test case steps in the Test Case editor or Test Report editor.)

For the **test case**, if the JSON response is configured (**Procedure Properties > Inputs and Outputs > Response**), then the response is available immediately in the **Response** view and the background is light grey (before running a test case). The purpose of the Response View is to make it easier to add analysis rules for test cases.

The purpose of the **Response View** is to make it easier to add analysis rules for test cases. See also "Defining a procedure" on page 223 in Chapter 10, "Procedures".

On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form.

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.
- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").

The **Response** view makes it easy to review responses and to copy part or all of a response for pasting into other documents. You will also use the Response view while creating analysis rules and while creating and working with response maps and form maps.

Notes:

- You cannot add an analysis rule, response map, or form map while viewing an unfiltered response.
- If you select more than one captured item, then the Response view becomes blank.

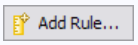
- The Response view does not display Web or Swing targets directly. To view the target associated with the response, click **Details**. You can view targets in Capture reports and in Test reports.
- To improve performance, iTest does not map all items in very long responses. If you notice that the "blue boxes" do not appear in the later text of a response, you can increase the setting so that iTest evaluates more queries. Click **Window > Preferences**. In the **iTest** group, go to **Response Mapping** and increase the **Maximum number of queries to evaluate** setting.
- You can copy/paste an entire test case document (for example, to create a test case that is very similar to the original). If you copy a test case, remember that the most recent response to each step (in the most recent execution of the test case) is also copied. This can result in old information in the Response view as you start to work on the "copy".

Response view context menu (right-click)

Copy	Copy the selected text onto the clipboard. Useful when creating response maps.
Select All	Select all response text.
Open Response Map	In the Response Map editor, open the response map that was used to identify the selected token.
Copy Query	Onto the clipboard, copy the query associated with the selected token.
Copy XPath	Onto the clipboard, copy the XPath query associated with the selected token. When you use the query, remember that iTest supports all XPath 1.0 functions.
Analysis Rule Wizard	Start the Analysis Rule wizard. (You must first select the step to apply the rule to.) See "About the Analysis Rule wizard" on page 667.
Quick Analysis Rule	Specify a basic analysis rule. See "Using the Quick Analysis Rule feature to specify an analysis rule for a step" on page 644.

Response view toolbar

	**View Unfiltered/Filtered response**: Toggle between viewing the filtered response and the unfiltered response. By default, the view displays the filtered response.
	**Toggle Sidebar**: While working on a response map in the Response Map editor, display or hide the sidebar that acts as a shortcut for response mapping tasks.
	**Add This Sample (as an Additional Sample) to an Existing Response Map**: Once you specify the response map in a dialog box, iTest adds the new sample and selects it on the **Samples** page of the Response Map editor. **Tip** This is a quick way to add a sample response for use in an emulated step. For a full discussion of implementing a virtual testbed by emulating responses, see Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions".
	**New Response Map**: Using the data for the current response as the sample, create a new response map.
	**New Form Map**: For Swing and Web snapshot steps, click New Form Map to start the Form map wizard. See "Overview: Form Maps" on page 843.

	Add a new analysis rule by starting the Analysis Rule wizard. (You must first select the step to apply the rule to.) See "About the Analysis Rule wizard" on page 667.
	View the **Action**, **Target**, and **Command** for the step. Useful for Swing and Web snapshot steps.
	On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form. • If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print. • If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly. Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**. **Note** When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").

Response view operation while developing test cases

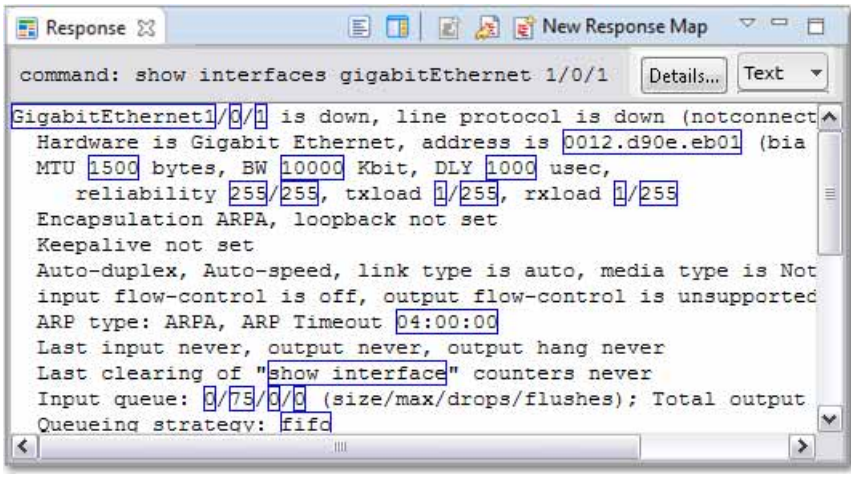
The easiest way to add an analysis rule to a test case step is to select the step in the Test Case editor, right-click in the appropriate blue box in the Response view, and then select **Insert Analysis Rule Using Wizard**.

Tip You can add a rule only if the step is selected in the Test Case editor. You cannot add a rule if the step is selected in the Test report editor.

Response view operation in the iTest Response Mapping perspective

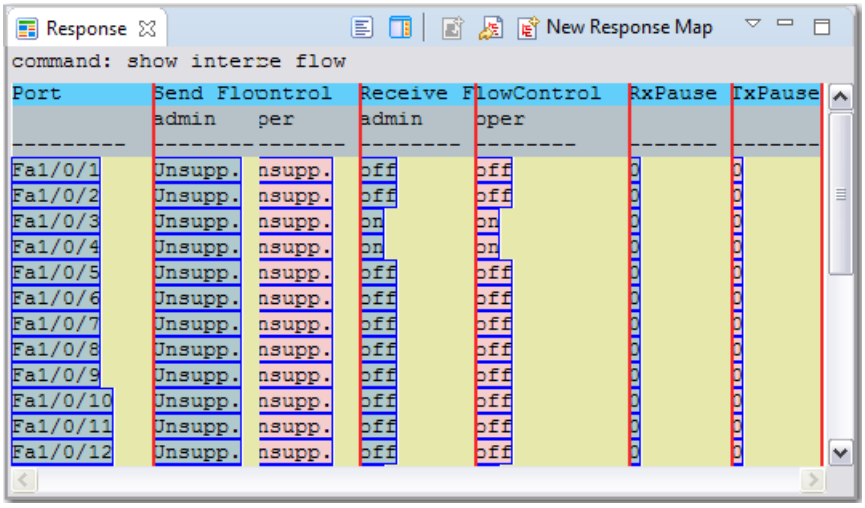
While you work in the Response Mapping perspective, the **Response** view places blue boxes around the values of identified tokens. The view updates the boxes whenever you make changes to token definitions.

When you select a token in a **Map** editor or a query in the **Queries** view, the associated value in the **Response** view is highlighted. In this example, we selected the MTU query in the Queries view, so the value of that token is highlighted.



Response view operation while working with Table response maps

While you work on a Table response map that uses character counts to determine column boundaries (the **Positional** setting), the Response view displays column markers that indicate the end of each column of data. To change the location of a column boundary, drag the marker to place it after the last character in the column, as shown in the example.



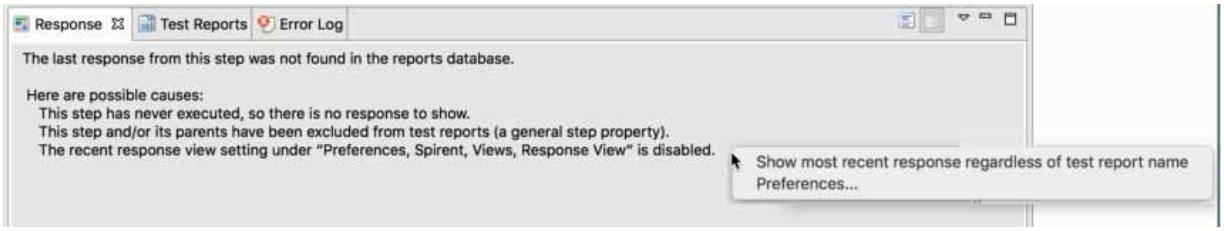
Response view operation while working in the Test Report Comparison editor

On the he **Comparison** page of the **Test Report Comparison** editor, when you select a step in the **Executed Steps Comparison** section, the Response view displays the reference response on the left and the target response on the right, separated by the | character. The Response view highlights in yellow the lines in the response that differ between the reference report and the target report.

Viewing the latest response view

iTest populates the response view from the most recent test report resulting from the test case execution.

For example, if you accidently run the QuickCall libraries and then run the test case that calls that QuickCall library, the QuickCall command step response of the test case displays empty with the probable causes. See below.



This is because iTest is reads from the most recent test report generated from the QuickCall library execution. To display the correct response from the Test Case execution, **Righ-click** and select the option to show the most recent response regardless of the test report name.

You may also delete the QuickCall test report, and then click on a step in the QuickCall library, the response view shows the expected response from the test case execution.

Note A QuickCall procedure with JSON Response, inserted in a Test Case populates the **Response View** with the JSON response from the called procedure and the Response View background is light grey before running a test case.

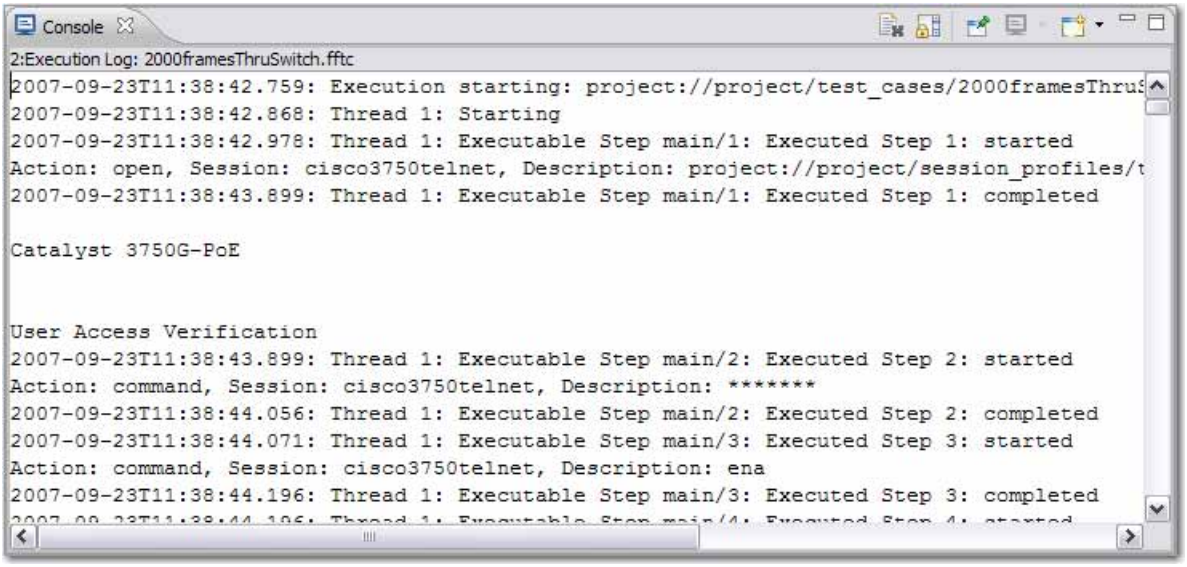
Console view

The Console view displays the real-time text log of test case execution. There are two Consoles; One for SNMP Traps and one that supports all other session types. The view displays the following types of information:

- Execution starting and ending
- Thread starting and ending
- Step execution started and completed
- Executable step information
- Executed step information
- Execution issues as they are declared

iTest-generated information that complements the data, such as timestamps, thread IDs, and step summary information appears in black or gray. Step execution (terminal log) information appears in blue.

You can open multiple instances of the console.



Console view toolbar

Clear Console	Clear all data from the Console view.
Scroll Lock	Lock the view to the current data so that the text does not scroll out of view. Data is being added, but scrolling is temporarily disabled.
Pin Console	Not implemented.
Display Selected Console	Display the data from the selected Console view.
Open Console	Select a particular type of console view to open.

Error Log

The Error Log displays exceptions. In the iTest Activities perspective, the log comes to the front when an exception occurs.

When working with Customer support, you can export and then email the contents of the log to help to resolve the issue.

Error Log toolbar

	Export log data to an Error Log text file (.log filename extension)
	Import log data from an existing Error Log file
	Clear the Error Log view
	Delete the Log data.

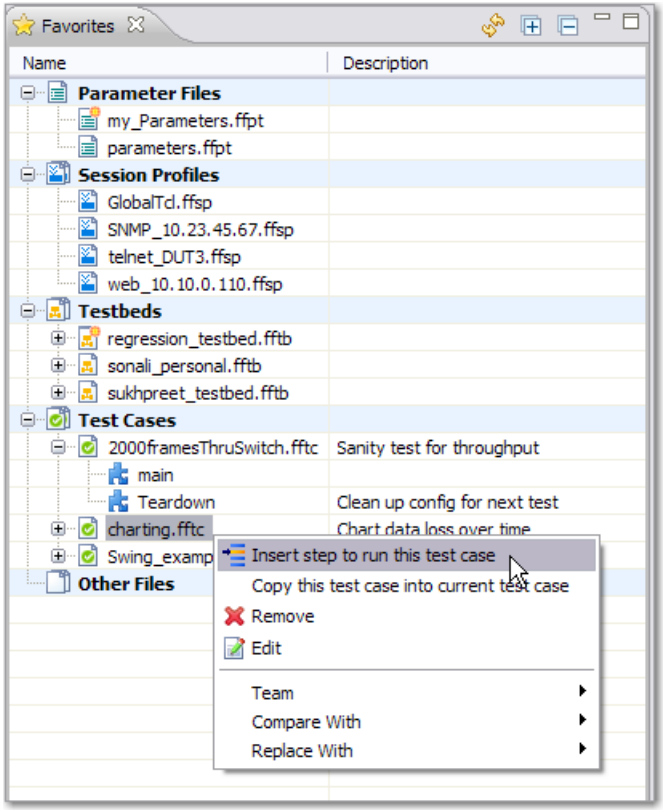
	Open an existing Error Log file
	Restore the Error Log to the data before the last deletion.

Favorites view

The Favorites view gives you ready access to

- Parameter files
- Session profiles
- Topologies and the devices in each topology
- Testbeds and the devices in each testbed
- Test cases and the procedures defined in each test case
- Other files that you specify

You can place any sort of file in your Favorites view for easy access. You can even add non-Spirent files (such as spreadsheets or word-processor documents) to Favorites.



- The parameter files, session profiles, topologies or testbeds, and test cases that you often use are easy to access from the Favorites view.
  - Double-click a testbed device or session profile to start a session.
  - Double-click any other type of iTest document to open it in the appropriate editor.

- Click  to refresh the view.
- Global parameter files, topologies, and testbeds are identified by the yellow star in the upper right corner of the icon  .
- The **Headline** text that appears here is based on the text in:
  - **Devices**: The **Description** property on the Testbed editor's **Devices** page. If there is no comment, then the session type appears
  - **Form maps**: The **Headline** property on the Form Map editor **General** page
  - **Procedures**: The procedure's **Headline** property value
  - **Response maps**: The **Headline** property on the Response Map editor **Overview** page
  - **Session profiles**: The **Headline** property on the Session Profile editor **Misc** page
  - **Testbeds**: The **Headline** property on the Testbed editor **General** page
  - **Test cases**: The **Headline** property on the Test Case editor **General** page
  - **Topologies**: The **@Name** attribute on the Topology editor **General** page
- Click  for a topology or testbed to view its devices.
- Right-click one or more devices and click **Start** to start interactive sessions.
- Click  to view the procedures that are defined in a test case.
- Right-click a test case or procedure to add steps to the test case that is currently being edited, as described in "Using the Favorites view to add steps to a test case".

Favorites view toolbar

 **Expand All**	Expand all items in the view.
 **Collapse All**	Collapse all nested items in the view.

Adding files to the Favorites view

Files listed in the Favorites view are actually shortcuts to the files (the files remain in the original folder).

- To add an item, drag it into the Favorites view.
- Alternatively, right-click it in iTest Explorer, and then select **Add to Favorites**.

For devices, if you do not specify a **Description** when defining the device on the **Devices** page of the Testbed editor, then the **Description** field in the Favorites view displays the session type for the device.

Removing files from the Favorites view

Right-click the item and select **Remove**.

Note Removing a file from the Favorites view **does not** remove it from the folder or workspace.

Opening (editing) files from the Favorites view

Double-click the filename in the Favorites view. The file opens as if you had launched it from iTest Explorer.

Note Rather than opening, a session profile will start a session when you double-click it in the Favorites view.

Starting a session using a session profile

By default, when you start a session by double-clicking a session profile from the Favorites view, or by right-clicking and selecting Start in the iTest Explorer, or Session Profile editor (the **New Session** page), the session starts in a new window.

Advanced Users Many of the property settings for session profiles support field replacements to enable you to parameterize settings so they can be determined dynamically at runtime. You might use **tcl**, **param**, or **profile** command field replacements to improve the flexibility and portability of automated test cases. Sometimes, to perform an interactive test, you might need to manually start a session that typically starts only for automated test sessions. To enable you to do this, if any **tcl**, **param**, or **profile** command field replacements are encountered while starting the session, iTest starts a Tcl interpreter so that the field replacement can be resolved.

When the session ends, the Tcl interpreter is disposed. When the session is restarted by pressing Enter, the substitutions are not made again. If a Tcl interpreter service is requested on restart, however, a new interpreter will be created and returned.

Starting a session using a device

When you start a session using the **device** in the Favorites view and then save the session into a test case, you make the test case more portable because the resulting **open** step refers to a device rather than a session profile. In addition, the **Testbed** property for the test case is updated to refer to the corresponding testbed file. See "Overview: Creating a test case" on page 125.

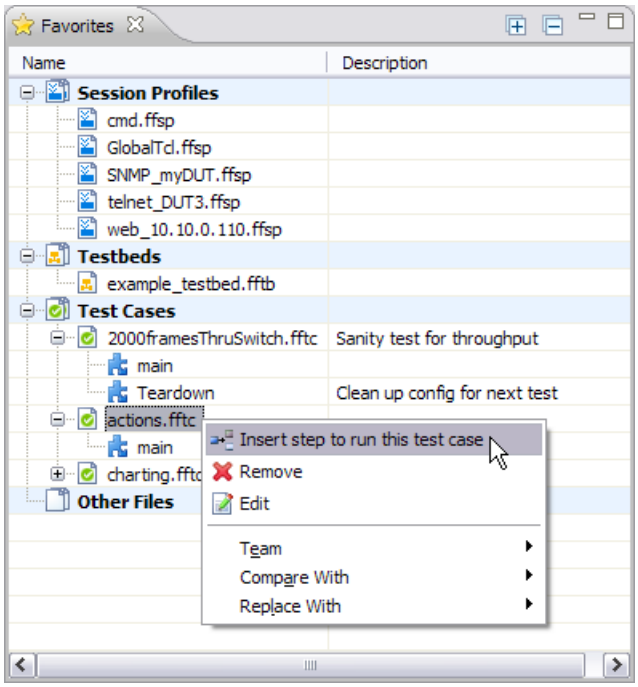
If the test case specifies a different testbed, then the **open** step refers to the session profile (or session type URI) that is specified in the **Inherits from** property for the device. The properties of the **open** step are taken from the device definition in the testbed file.

Using the Favorites view to add steps to a test case

If you're currently editing a test case (in the Test Case editor), you can use the Favorites view to quickly add steps, procedures, or entire other test cases to it.

- For a test case, click  to view the procedures that are defined in a test case.

- Right-click a test case or procedure to add steps, a procedure, or a full test case to the test case that is currently being edited.



- Right-click a test case and select **Insert step to run this test case** to add a **run** step that executes the favorite test case as a child (foreign) test case. The **run** step is added after the selected step in the test case being edited.
- Right-click a test case and select **Copy this test case into current test case** to paste all procedures and steps from the favorite test case into the test case being edited. If a procedure name already exists in the test, then iTest creates a unique procedure name.
- Right-click a procedure and select **Insert step to call this procedure** to add a **call** step that calls the procedure. The **call** step is added after the selected step in the test case being edited. Only
- Right-click a procedure and select **Copy this procedure into current test case** to add the full procedure into the test case being edited. If the procedure name already exists in the test case, then iTest creates a unique procedure name.
- Right-click a procedure and select **Insert procedure's steps into current test case** to add the steps of the current procedure after the selected step in the test case being edited. The procedure definition row is not added. If you had selected a procedure row in the Test Case editor, then the steps are added to the end of the procedure.

Creating a procedure call step from the Favorites view

Use this method to quickly add a call when a test case or procedure library appears in the Favorites view. "Creating a procedure 'call' step from the Favorites view" on page 233.

Images view: Saving window images

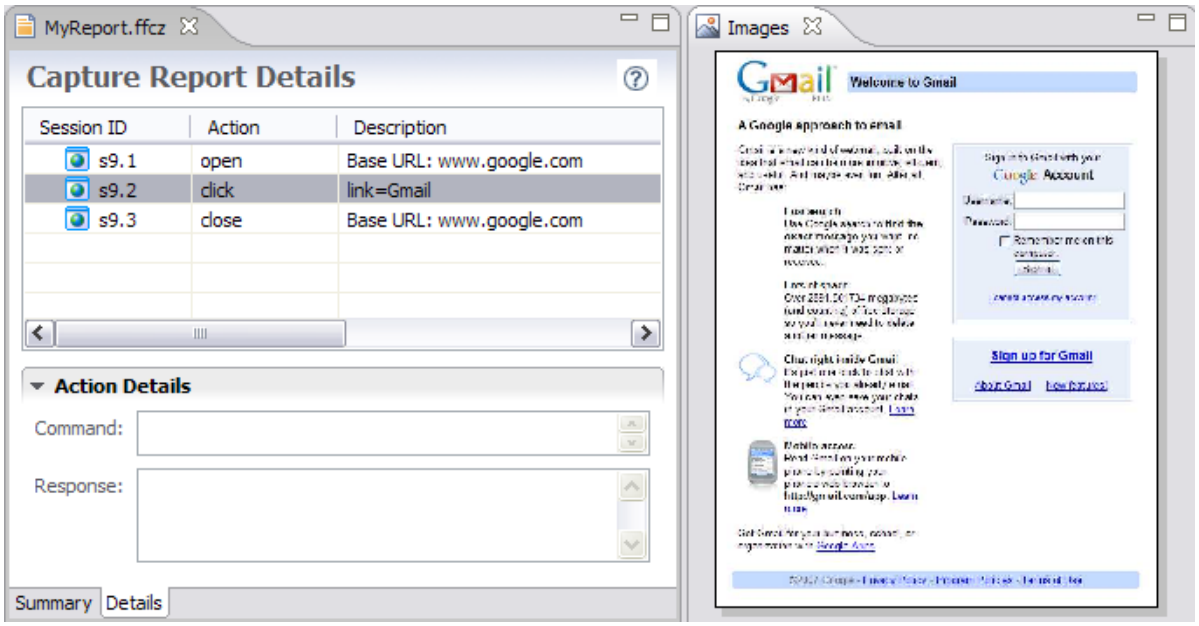
During browser-based sessions, various web pages appear in the browser. For reporting results or documenting a test, you might want to capture a snapshot of a page. The snapshot appears in the Images view.

During sessions with Swing applications, you might also want to capture a snapshot of a window.

For example, you can take a snapshot of a page at any time by clicking **Snapshot** in the session window.

Viewing snapshots

To view the snapshot of a page, select a captured item in either the Capture view or the Capture Report editor (the captured item must be associated with a page load event). The snapshot appears in the Images view (**Window > Show View > Images**)



Problems view

iTest performs validation on your changes to documents. When you take action that has the potential to cause problems (for example, during execution), iTest opens the Problems view and posts a message into the view.

For example, in the Capture view, while saving steps from multiple sessions as a procedure, you might forget to include an **open** step for each session. This fact would then appear as an item in the Problems view

Progress view

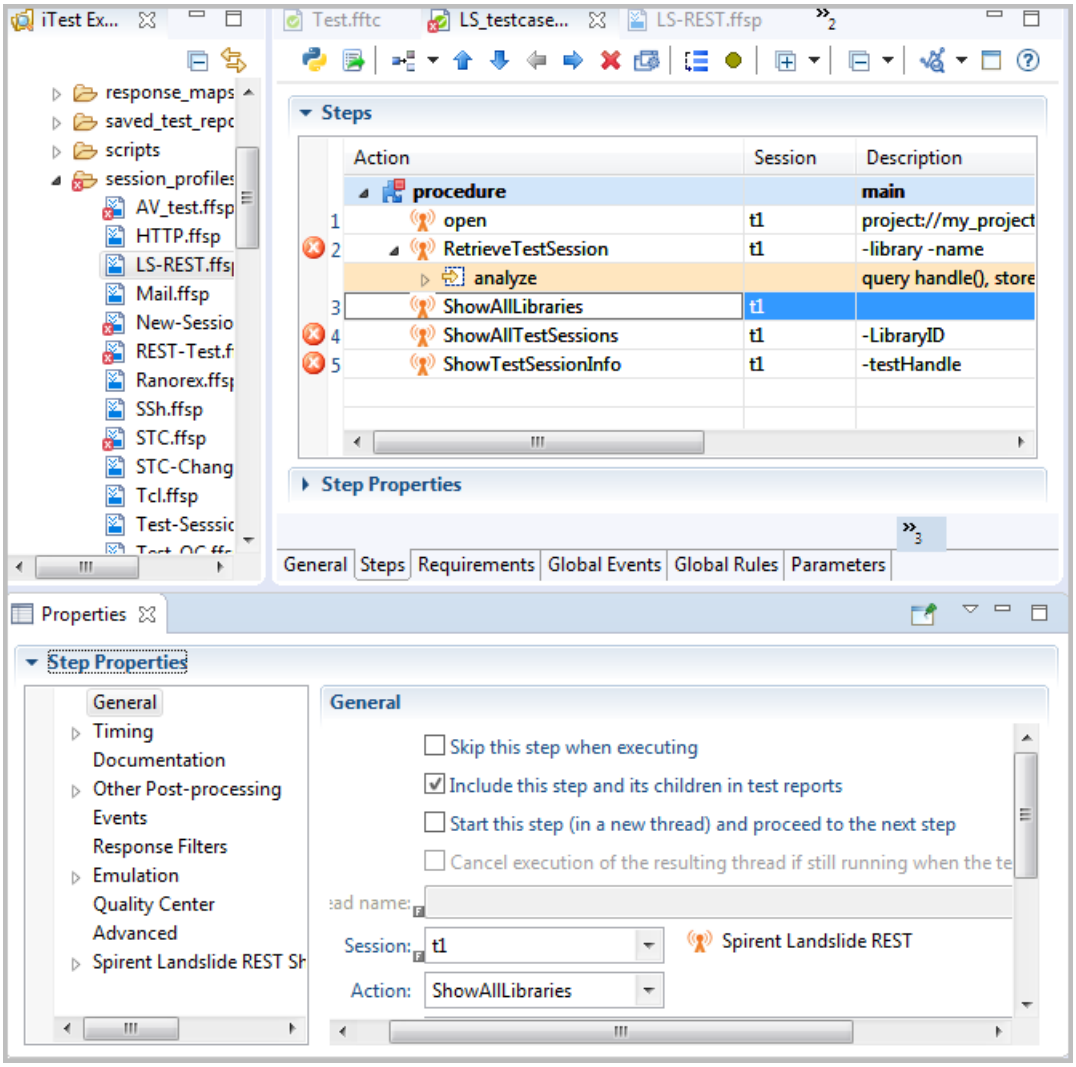
The **Progress** view displays a progress bar for background tasks (like loading the Capture view).

You will not typically use the **Progress** view, but you may find it helpful when a process takes longer than you expect

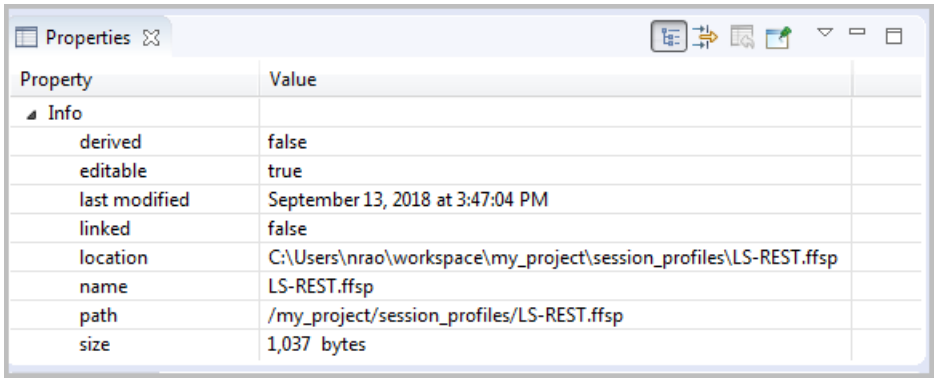
Properties view

The Properties view displays information depending on your selection, that is, whether you have selected a test case step or an item in the iTest Explorer (alphabetically). The example below shows a step in the Test Case editor and the Properties view lists all properties associated with the step.

Note A separate **Properties View** window displays if you have selected the **Show Properties View** option from the right-click menu. See "Show Properties View" on page 157 (Chapter 8, "Test Case Editor"). You may edit step properties using either the Step Properties section (within the Test Case Editor) or via the Properties View window.



Selecting an item in the **iTest Explorer** displays all its associated properties in the **Properties view**.



The Properties view takes on special functions while you work in the Topology editor. See "Layout of the Topology editor" on page 491.

Queries view

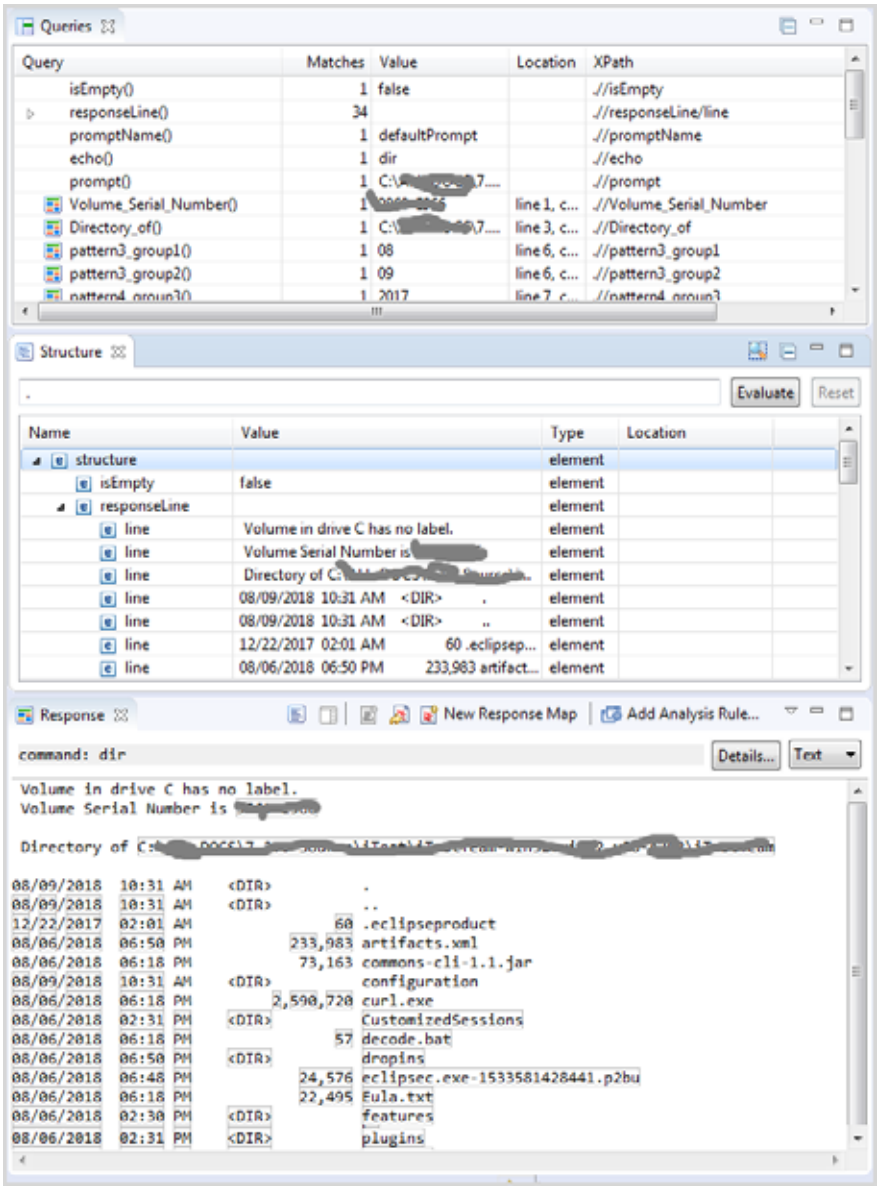
When iTest searches a response for a value that a test case step has requested, it uses an XPath query to search for the specified value in an XML-format structured data representation of the response text. XPath is the foundation on which response mapping is built.

iTest auto-maps query that matches the entire line of text on a response so that you do not have to manually enter a regular expression in your analysis rule. This auto-mapping of the query that matches all the lines of text on a response, assists you with creating your test cases faster.

For JSON object response, iTest auto-maps queries of for keys at the root level of a JSON object response. No auto-mapped queries are created for nested keys of a JSON object response.

The Queries view lists the XPath queries and their results for the response that is displayed in the Response view. (Queries can be defined in a response map or in local analysis rules. In addition, iTest auto-generates queries for structured responses like Web, SNMP, traffic generator devices, and XML.)

When you select a test case step the associated Queries view, Response view and the Structures view displays..



Note The name that appears in the **Query** column is the friendly name for the token query; the actual XPath query appears in the **XPath** column. iTest auto-generates names for queries. You can modify the names and add custom aliases on the Queries page of the Response Map editor.

The responseLine() query applies the regex [^\r\n]+ to extract entire linesfrom response:

- For a multi-line response, the contents of responseLine() will be a list of lines (xPath: .//responseLine/line).
- For a single-line response, the contents of responseLine() will be a response string (xPath: .//responseLine).

The responseLine() query (unlike other auto-response mappings), does identify the value of this response within a grey box. For example, if the response of a command as shown below.

```
latency_us: 10
throughput_gbps: 40
```

Only the integers 10 and 40 would be enclosed in grey boxes, corresponding to the auto-mapped `queries latency_us()` and `throughput_gbps()`.

In aditional, not all queries represent tokens in the response. iTest generates some other useful queries like **RowCount** for table maps.

Tip To improve performance, iTest does not map all items in very long responses. If you notice that the “blue boxes” do not appear in the later text of a response, you can increase the setting so that iTest evaluates more queries. Click **Window > Preferences**. In the **iTest** group, go to **Response Mapping** and increase the Maximum number of queries to evaluate setting.

Fields in the Queries view

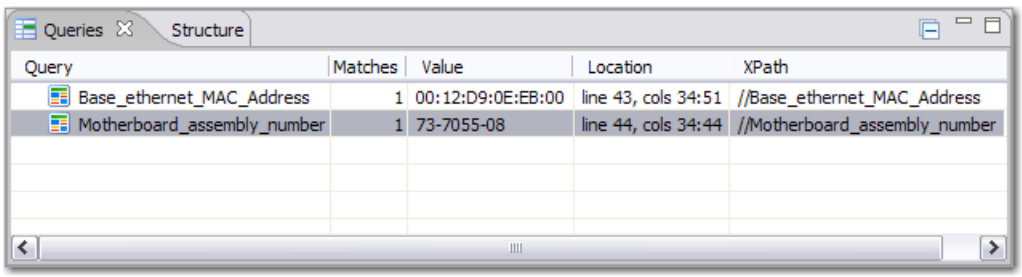
Query	Queries are friendly names for XPath queries and make advanced or conditional queries easier to read and more flexible. You can use queries when creating analysis rules in test cases. Queries are the token names that you define for Regex maps.
Matches	The number of matches in the response text that is currently in the Response view.
Value	The values of the matches in the response text that is currently in the Response view.
Location	The location of the match by line and column location in the response.
XPath	The actual query in XPath notation. When you use the query, remember that iTest supports all XPath 1.0 functions

Example: How an XPath query works

A portion of the response to the show version command might be:

```
Image text-base: 0x00003000, data-base: 0x00C7FC04
```

We have defined a Regex map with two named two tokens `Image_text_base` and `data_base`. The Queries view displays each of the named regex groups as a query that can extract a value from a response.



Here is a portion of the XML of the response map that abstracts the two hex value tokens:

```
<mapped>
     <Block id="com.fnfr.svt.mapping.block"/>
     <Regex id="com.fnfr.svt.mapping.regex">
       <Body>
         <regex1 map:endcol="50" map:line="5" map:startcol="0">
         <Image_text_base map:endcol="27" map:line="5" map:nodetype="token"
map:startcol="17">0x00003000</Image_text_base>
           <data_base map:endcol="50" map:line="5" map:nodetype="token"
map:startcol="40">0x00C7FC04</data_base>
         </regex1>
       </Body>
     </Regex>
     <Tabular id="com.fnfr.svt.mapping.table"/>
  </mapped>
```

The XPath query used to reference the data_base token is:

```
//data_base
```

The query returns everything contained between the `<data_base>` and `</data_base>` tags (in the sample response, the value `0x00C7FC04`).

Note You cannot extract value of a hierarchical data structure in original response syntax. You can to get joined data of that structure. For exampe, if you have the following python response:

```
{
"key": [1, 2 ,3]
}
```

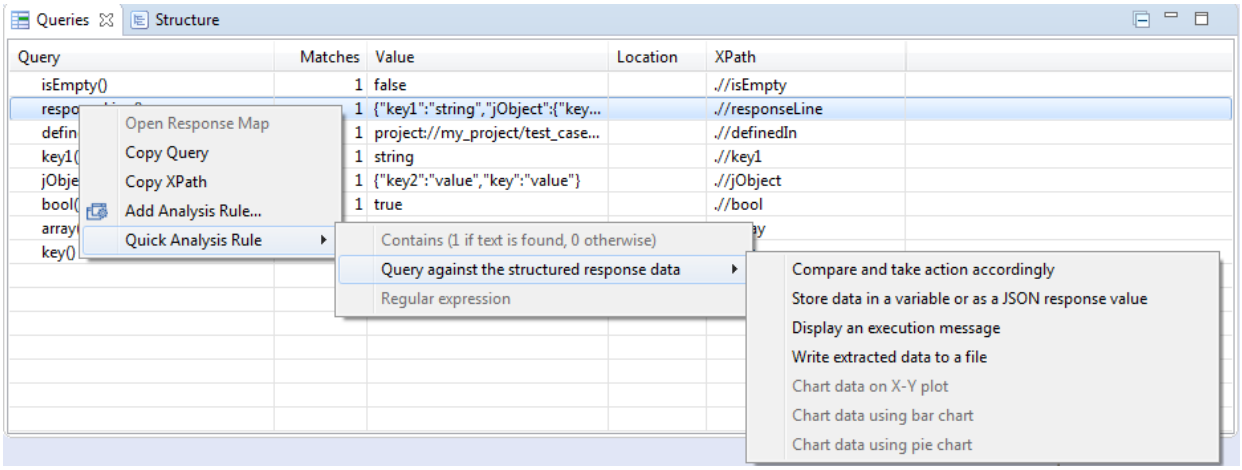
Then you can get the value of list by the following:

```
XPath: mapped/Python/item/item[1]/item.
```

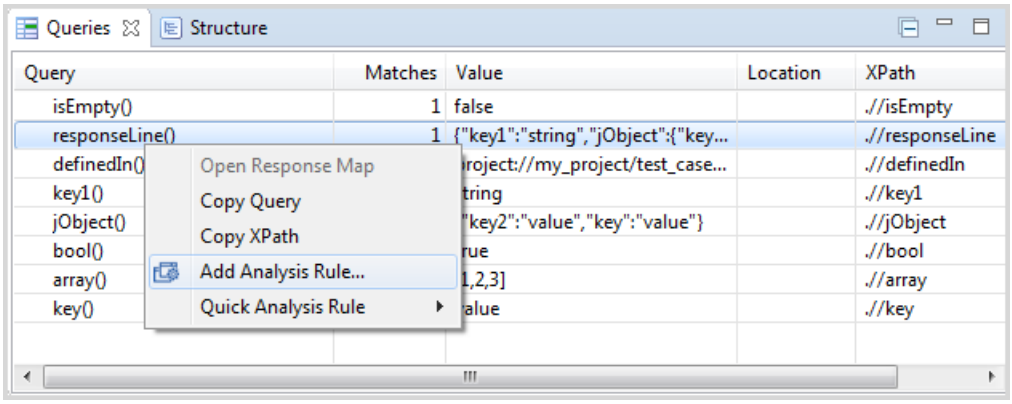
But value will be “123” and not [1, 2, 3].

Adding an analysis rule from the Queries view

While working in the Test Case editor on a step for which a response map is specified (for the Expected Response): You can right-click a query in the Queries view to add an analysis rule that uses the query to extract the value (that is, use the query as the extractor) and specify the processor (what to do when the match is returned — ‘Compare and take action’ in the example).



You can also launch the Analysis Rule wizard:



Queries view: While working in the Test Report Comparison editor

On the **Comparison** page of the **Test Report Comparison** editor, when you select a step in the **Executed Steps Comparison** section that has a response map associated with it, the Queries view identifies differences between query results using the diff icon . For details, see "Examining queries for differences" on page 446.

About XPath functions

iTest supports all XPath 1.0 functions.

For an XPath language description, see **http://www.w3.org/TR/1999/REC-xpath-19991116**

iTest supports the following XPath functions: **http://jaxen.org/apidocs/index.html**

Search view

When you use **Search** to locate files with a particular filename or that contain particular text, the Search view displays the results of the Search operation.

Toolbar

The toolbar in the Search view contains the following buttons:

Show Next Match	This command highlights the next match of the search expression in the editor area, opening the file if required.
Show Previous Match	This command highlights the previous match of the search expression in the editor area, opening the file if required.
Remove Selected Matches	Removes all highlighted matches from the search results.
Remove All Matches	Removes all search result from the search view
Expand all	Expands every tree item in the search view
Collapse all	Collapses every tree item in the search view
Run the Current Search Again	This command reruns the current search again, so that removed search results reappear or changes are reflected.
Cancel Current Search	Cancels the current search

Show Previous Searches	This command allows you to browse previously conducted searches and repeat a previous search. You can select a previous search from the drop-down menu or clear the search history.
Pin the Search view	Pinning the search view means that subsequent searches will shown their results in another search view and that the pinned view remains unchanged.

Step Issues view

While you work on a response map or test case, the Step Issues view shows Errors, Warnings, and info messages associated with queries on the response in the Response view. The Step Issues view lists and describes unexpected items in the current sample, enabling you to "test drive" responses to determine whether named tokens (queries) work as you expect.

Click an issue to view the associated item in the Response, Structure, and Queries views.

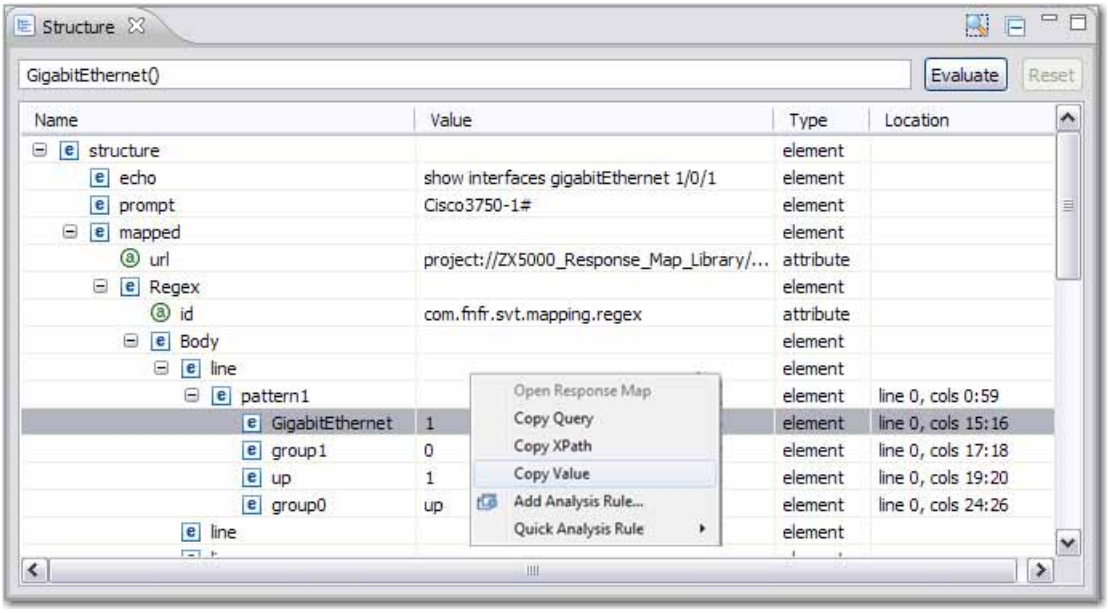
Originator	The map or line in the sample response in which iTest has detected the issue. The Issues view groups the issues into Errors, Warnings, and info Messages — these group titles also appear in the Originator column at the head of each group.
Message	Description of the issue, including execution messages that are generated by analysis rules. If you use the Analysis Rule wizard to auto-generate a message, the actual text of the resulting message appears here. "Pass" identifies rules that set the test case **Result** to **Pass**.
Location	The location in the response by line and column location of the issue.

Tip You can copy one or multiple issues to the clipboard. Select (Ctrl+click or Shift+click for multi-select) and then press Ctrl+C (or right-click and select **Copy)**.

Structure view

iTest stores a structured XML representation of every response. Use the Structure view to view the structured part of the response. Click **Viewing Mode** to switch between the following view formats:

- A table representation of the XML data. The response structure is represented by the tree structure in the **Name** column. In the example, we have selected an element named **GigabitEthernet**.

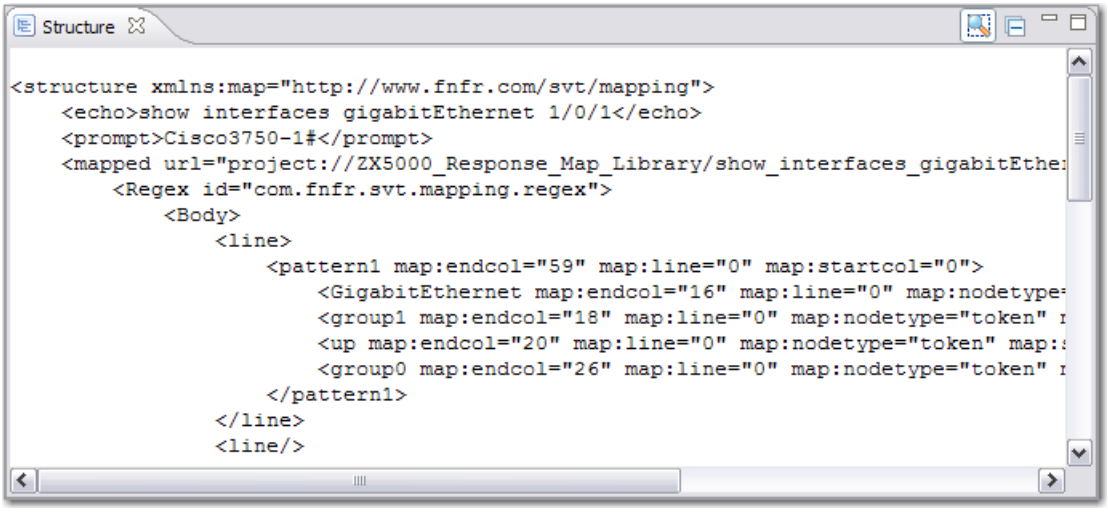


You may select a row and copy the value of the response Structure to the clipboard and paste it into another area in your test case editor. Select a row in the Structure view, Right-click, then select **Copy Value** to copy the value (e.g., 1) and paste it anywhere in the editor. If the selected row does not contain any value, the **Copy Value** option is disabled.

Structure view columns

Name	The name of a node in the XML structure.
Value	The text contained by the selected node.
Type	**element, attribute:** The type of node or value returned by evaluating the expression in the text box.
Location	The location of the match in the response by line and column location.

- XML representation of the response. The XML document displayed in the Structure view is the basis for identifying tokens and targets.

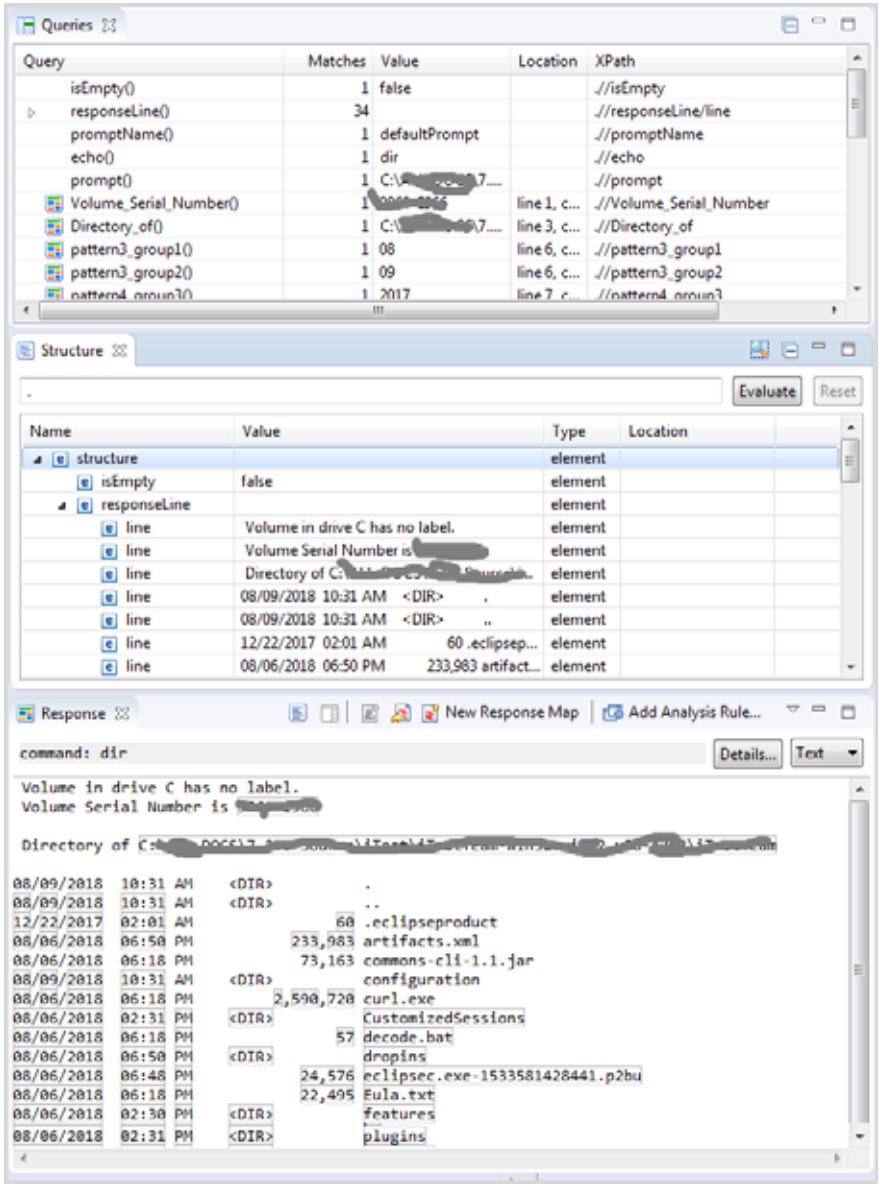


Structure view toolbar

	Switch between viewing modes: • XML text • Structured representation of the XML
	In the structured viewing mode, collapse all indented items into the top-level headings.

Relationship between the Structure, Response, and Queries views

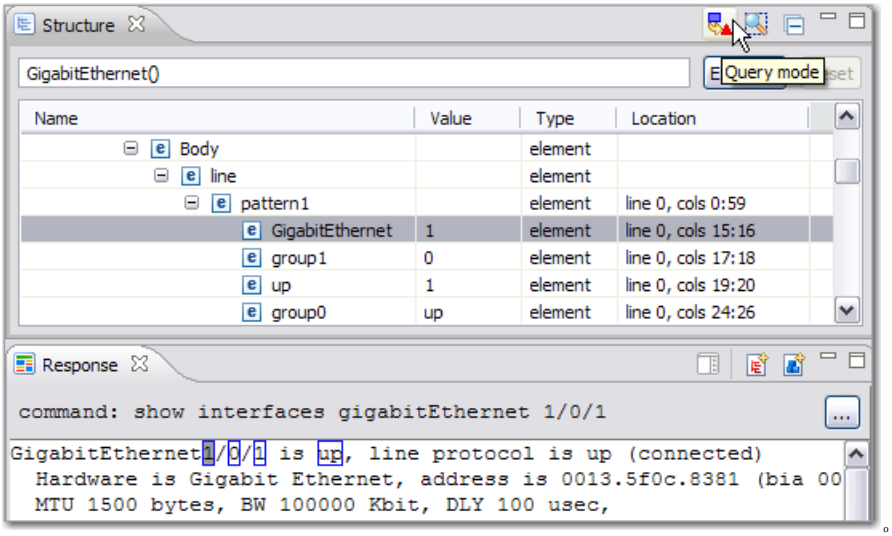
When you select a query in the Structures view, the associated matching text is selected in the Response view and in the Queries view.



Working with query definitions

While viewing the response structure in the table format, the text box enables you to edit query definitions. As you edit an XPath query, the view updates to highlight appropriate matches in the structured part of the response. Remember that iTest supports all XPath 1.0 functions.

1 As we begin, the query appears in iTest “friendly name” format.

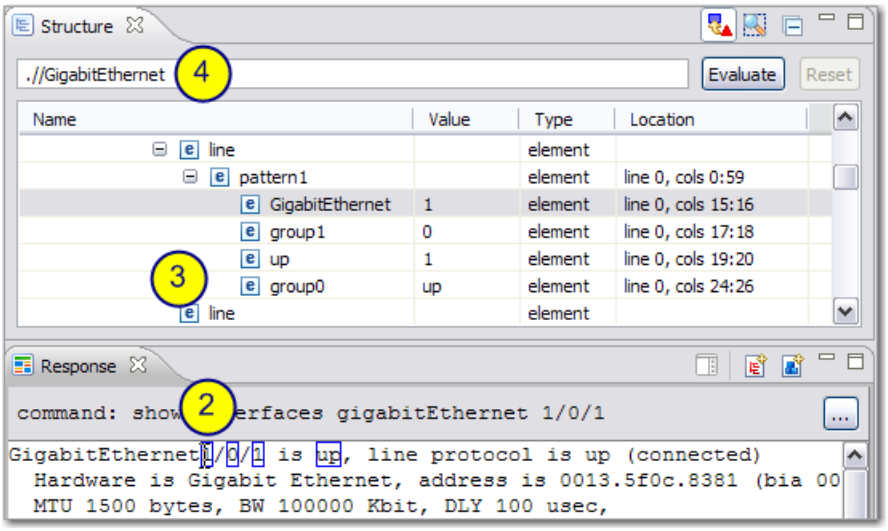


2 In the Response view, click in another token (**group0 — up**, in this example).

3 Now click in the original token again.

4 Notice that the original query now appears in XPath form.

You can now edit the query as needed to help you to develop a custom query (remember that iTest supports all XPath 1.0 functions).



Evaluating queries

Once you have edited a query to your satisfaction, click **Evaluate**. The following events happen:

- The full XPath query appears in the text box
- All matches appear in the Structure list
- Click **Reset** to reset the view to its default appearance.

For Response Maps:

Each named token appears in a structure that reflects the token's location in the Response map and its name, value, and other defining characteristics. Whenever a test case performs a mapping query, the query is applied against the XML document.

While executing the test case, analysis rules check the document to compare values, perform analysis, and so on.

Tokens extracted using RegEx token mapping are identified by the **<regex>** XML tag.

For TL1 Response Maps:

See "Mapping TL1 responses" on page 1216.

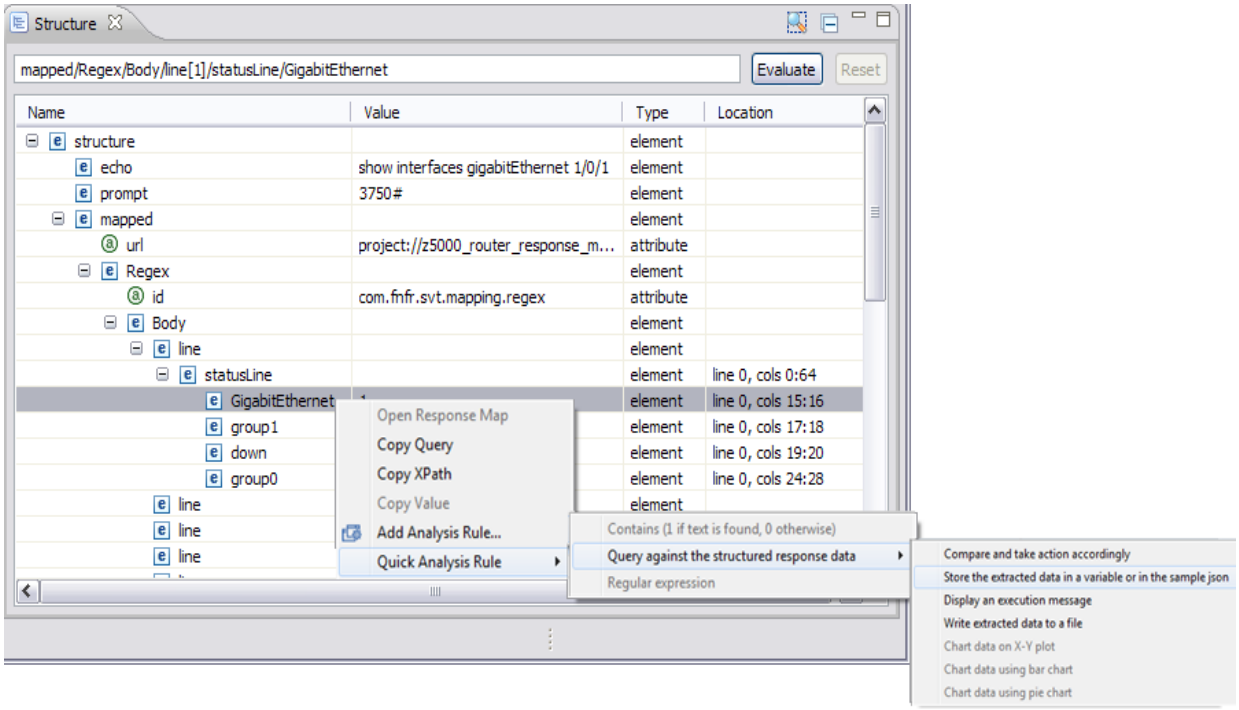
For Form Maps (HTML and web steps only):

The Structure view displays the structured XML representation of the form (web page). This XML document is the basis for identifying targets on the form (elements like buttons, links, and fields on the page). Targets are XPath queries that uniquely identify the buttons, links, or fields on the page that the step takes action on.

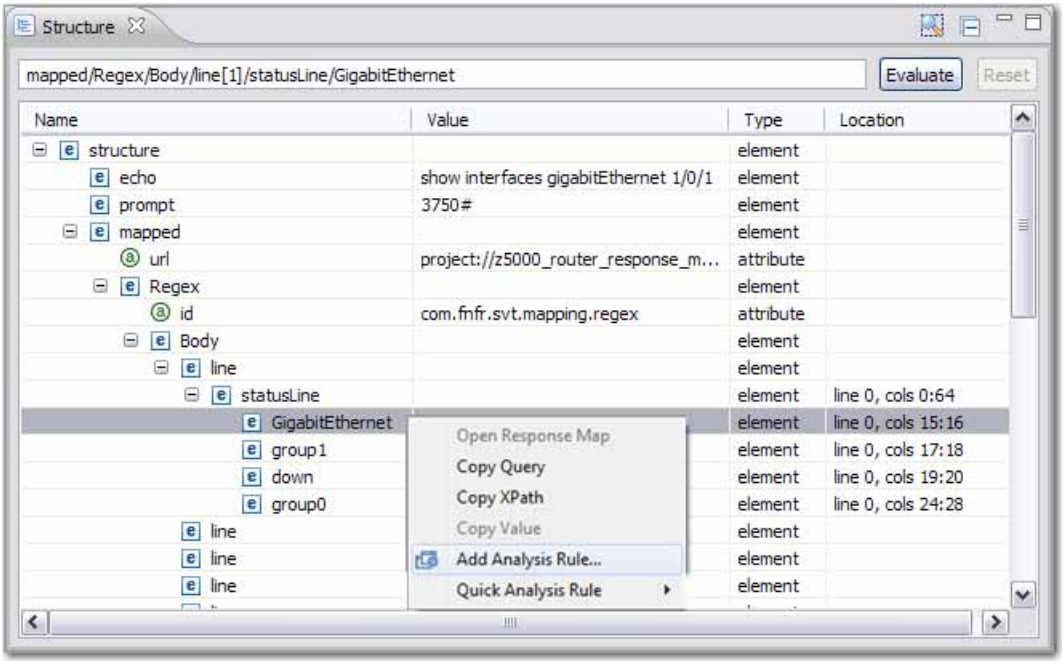
So, the form map is the *Context* of a test case step, and you specify an ***Action*** to perform (click, type text, and so on) on a ***Targets*** (a button, link, field, and so on).

Adding an analysis rule from the Structure view

While working in the Test Case editor on a step for which a response map is specified (for the Expected Response): You can right-click an element in the Structure view to add an analysis rule that uses the associated query to extract the value (that is, use the query as the extractor) and specify the processor (what to do when the match is returned — Save into a variable in the example).



You can also launch the Analysis Rule wizard:



About XPath functions

iTest supports all XPath 1.0 functions

For an XPath language description, see

http://www.w3.org/TR/

iTest supports the following XPath functions:

http://jaxen.org/apidocs/index.html

Setting Preferences for Views

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”. The following shows setting preferences for capture, error long, execution, and respnse views.

Setting preferences for capture view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > View >Capture View** . The **Capture View** window opens, set up the capture view as required.

Display replay tips in the capture view	Default: Selected. Select to indicate that iTest should display replay tips in the capture view. Manual step: Manual steps will be displayed in the color selected. You may pick a custom color to display manual step as follows. Click the displayed color and select the required color from the color palette.

Setting preferences for error log view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > View >Error Log View** . The **Error Log View** window opens, set up the error log view as required.

Open the Error Log View on any error, warnings, or exceptions	Default: Not selected. Select to indicate that iTest should open the Error Log view when any error, warning, or exception occurs.

Setting preferences for execution view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > View >Execution View** . The **Execution View** window opens, set up the execution view as required.

Display execution speed slider in the Execution View	Default: Not selected. Select to indicate that iTest should populate the contents of the response view based on the most recent execution of the step selcted, regardless of the test report name.

Setting preferences for response view

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > View >Response View** . The **Response View** window opens, set up the response view as required.

Show most recent response regardless of test report name	Default: Not selected. Select to indicate that iTest should populate the contents of the response view based on the most recent execution of the step selcted, regardless of the test report name.

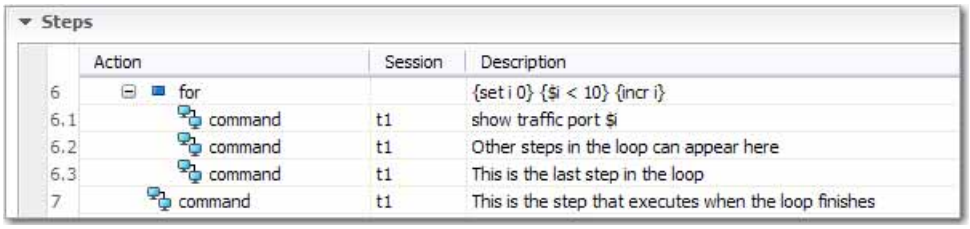
CHAPTER 16

Controlling execution flow: Loops, If/Then, and Switch

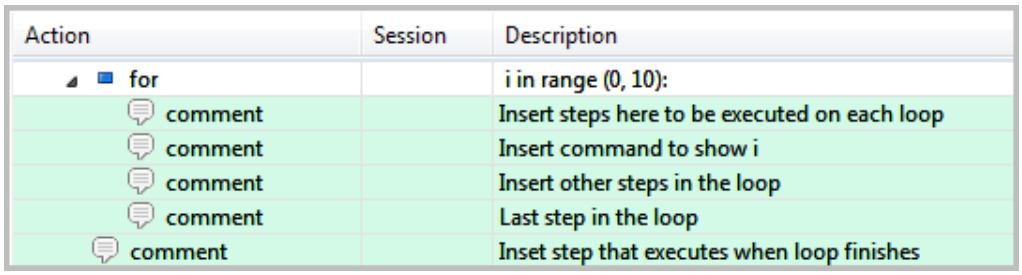
Overview: Loops and flow-control logic

The for loop

A **for** loop executes a group of steps a specified number of times. For example, repeat the steps within the loop 10 times.



Python:



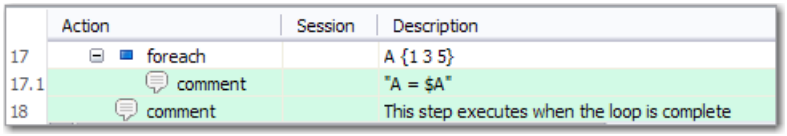
See “The for action: Execute a group of steps in a loop” on page 321 for instructions on creating a **for** loop and a detailed description of how **for** loops operate.

The foreach loop

A **foreach** loop performs the steps within the loop for each value in a specified set of values.

Example: foreach A {1 3 5}

In this example, the first list has a single member, A. The following steps result in three **comment** steps being executed with A = 1, A = 3, and A = 5:



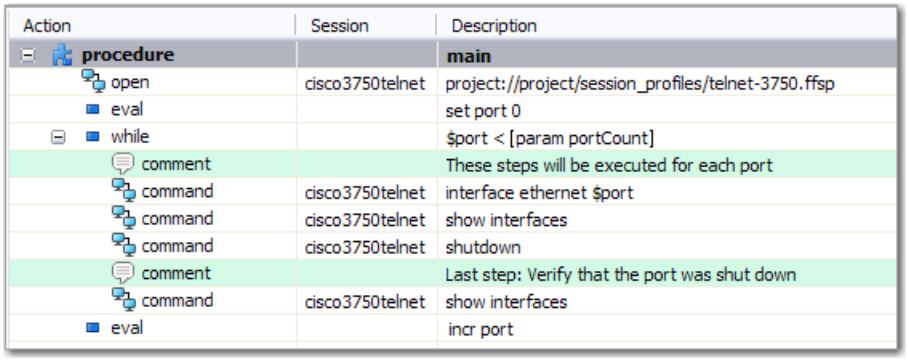
	Action	Session	Description
17	foreach		A {1 3 5}
17.1	comment		"A = $A"
18	comment		This step executes when the loop is complete

See “The foreach action: Execute a group of steps in a loop” on page 324 for instructions on creating a **foreach** loop and a detailed description of how **foreach** loops operate.

The while loop

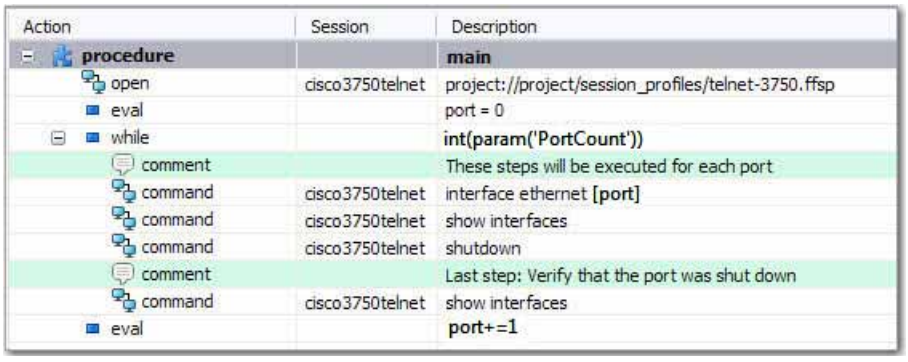
A **while** loop repeats a group of steps until a specified condition is no longer true. The **while** loop consists of a group of steps and a condition. The condition is first evaluated. If the condition is true, then the steps are executed. This repeats until the condition becomes false.

For example in **Tcl**, **while $port < [param portCount]** means: While the value of the **port** variable is less than the number of ports on the card (the value of the portCount parameter), repeat the loop. As soon as the **port** number is greater than or equal to **portCount**, leave the loop and continue executing at the first step after the **while** loop.



Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		set port 0
while		$port < [param portCount]
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet $port
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		incr port

For example in **Python**, **while port < int(param('PortCount'))** means: While the value of the **port** variable is less than the number of ports on the card (the value of the portCount parameter), repeat the loop. As soon as the **port** number is greater than or equal to **portCount**, leave the loop and continue executing at the first step after the **while** loop

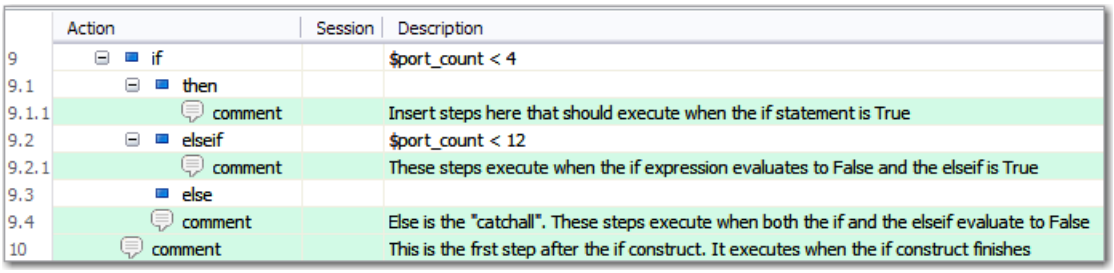


Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		port = 0
while		**int(param('PortCount'))**
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet **[port]**
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		**port+=1**

See “The while action: Repeat the steps in a ‘while’ loop” on page 325 for instructions on creating a **while** loop and a detailed description of how **while** loops operate.

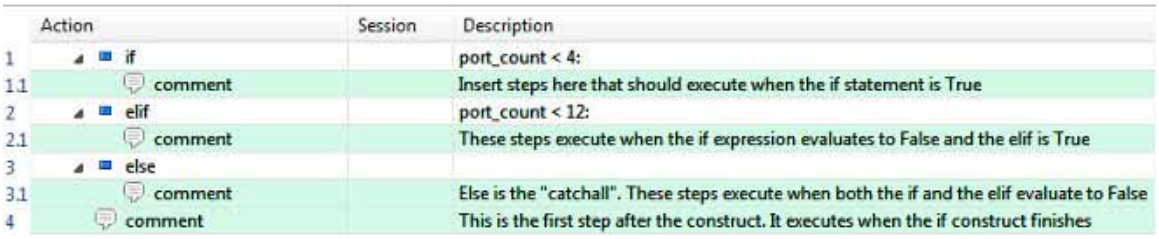
The if / then construct

An **if** step evaluates the expression that appears in the **Description** cell (the value of the **Command** property).



	Action	Session	Description
9	if		$port_count < 4
9.1	then		
9.1.1	comment		Insert steps here that should execute when the if statement is True
9.2	elseif		$port_count < 12
9.2.1	comment		These steps execute when the if expression evaluates to False and the elseif is True
9.3	else		
9.4	comment		Else is the "catchall". These steps execute when both the if and the elseif evaluate to False
10	comment		This is the frst step after the if construct. It executes when the if construct finishes

Python:



	Action	Session	Description
1	if		port_count < 4:
1.1	comment		Insert steps here that should execute when the if statement is True
2	elif		port_count < 12:
2.1	comment		These steps execute when the if expression evaluates to False and the elif is True
3	else		
3.1	comment		Else is the "catchall". These steps execute when both the if and the elif evaluate to False
4	comment		This is the first step after the construct. It executes when the if construct finishes

If the expression is True, then continue execution at the immediately following step.

If the expression is False, then continue execution at the associated **elseif** action. If there is no **elseif** or **elif** action, or if the **elseif**/**elif** action evaluates to False, then continue execution at the associated **else** action.

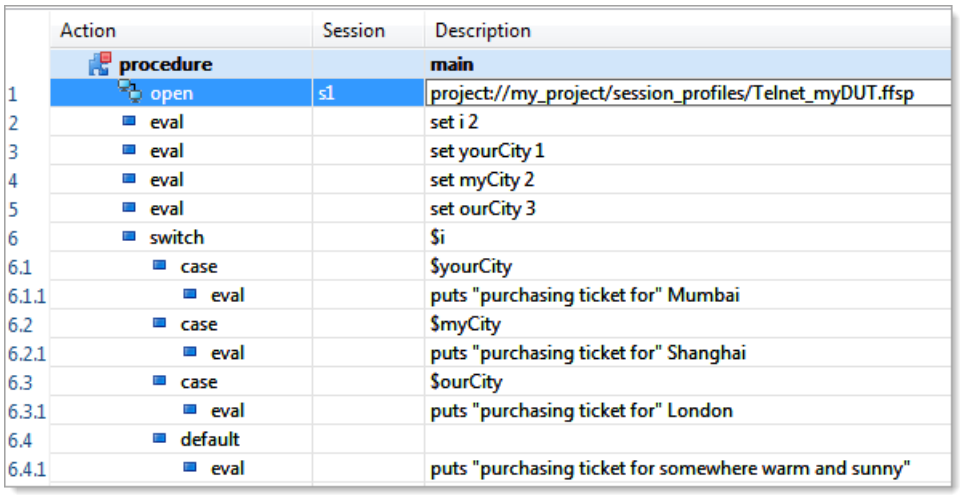
When the steps indented under the **else** are complete, then continue execution at the immediately following step (that is, after the last step in the **if** construct).

See “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329 for instructions on creating an **if-then-else-elseif** construct and a detailed description of how **if** constructs operate.

The switch construct

Note Switch construct does not apply to Python test cases

A **switch** construct allows the value of an expression to determine the flow of execution. See “The ‘switch’ action: Control execution flow based on the value of a variable or expression (Tcl)” on page 334.



	Action	Session	Description
	procedure		**main**
1	open	s1	project://my_project/session_profiles/Telnet_myDUT.ffsp
2	eval		set i 2
3	eval		set yourCity 1
4	eval		set myCity 2
5	eval		set ourCity 3
6	switch		$i
6.1	case		$yourCity
6.1.1	eval		puts "purchasing ticket for" Mumbai
6.2	case		$myCity
6.2.1	eval		puts "purchasing ticket for" Shanghai
6.3	case		$ourCity
6.3.1	eval		puts "purchasing ticket for" London
6.4	default		
6.4.1	eval		puts "purchasing ticket for somewhere warm and sunny"

Substitutions in the Command text

The command for **while**, **forEach**, **for**, and **if** actions is directed at the iTest interpreter. To ensure that iTest commands like [tcl] or [tclexpr] will be correctly interpreted, the text for the **Command** property (that appears in the **Description** cell) is interpreted as literal text. The property that controls substitution for the step is disabled (the **For the Command field, perform command, variable, and backslash substitution** property).

As a result, the text is not processed for the following substitution types before the step is executed (substitution occurs during execution):

- Command field replacements (for example, **char**, **expr**, **param**, **query**, or **response**)
- Variables
- Backslash characters used to escape special characters

Inserting for, foreach, if, switch, and while constructs into a test case

Note Python supports only **for**, **if**, and **while** loop constructs.

The easiest way to add program control logic is to follow this procedure:

1. Create the steps that should appear within the construct (the steps must be a contiguous group).
2. Select all the steps (use Shift-click).

3 Right-click, and then select **Wrap in** the type of construct (for example, **while statement**).

(Alternatively, use the **Alt-Shift-w** keyboard shortcut)

4 iTest adds the appropriate control step (**if**, **for**, **foreach**, **switch**, or **while**) and then indents the selected steps to indicate that they are included in the construct.

- **If** statements wrap the selected steps in a **then** step and include an **else** step after the selected steps. You have the option to add an **elseif** step if needed.
- **Switch** statements wrap the selected steps in a **case** step and include a **default** step after the selected steps.

5 Now edit the control step as described in the appropriate section (for example, to set the number of repetitions in the **for** loop).

"The for action: Execute a group of steps in a loop" on page 321

"The foreach action: Execute a group of steps in a loop" on page 324 **(Tcl only)**

"The while action: Repeat the steps in a 'while' loop" on page 325

"The 'if' action: Element of an if/then or if-elif-else construct" on page 329

"The 'switch' action: Control execution flow based on the value of a variable or expression (Tcl)" on page 334 **(Tcl only).**

For and ForEach loops

Note Python supports **for** loop construct and not **ForEach**.

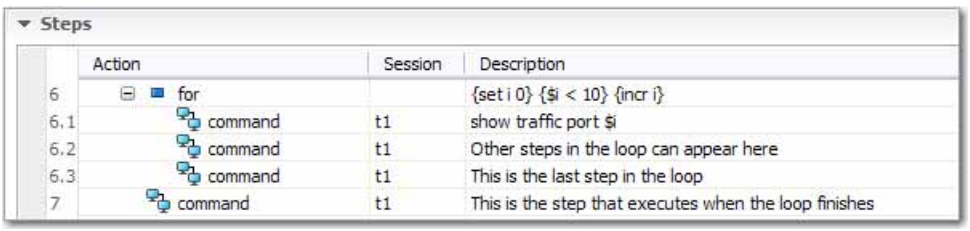
The for action: Execute a group of steps in a loop See page 321.

The foreach action: Execute a group of steps in a loop See page 324.

The for action: Execute a group of steps in a loop

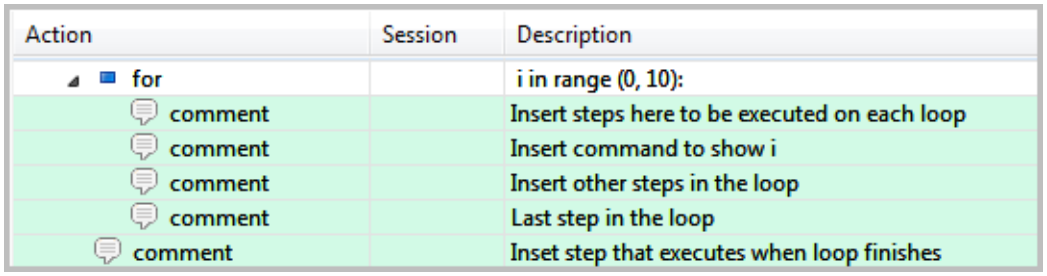
A **for** loop executes a group of steps a specified number of times. For example, repeat the steps within the loop 10 times. A **for** loop is composed of all of the steps that are indented under the EXEC **for** clause:

Tcl example: The **Description** cell (the value of the **Command** property) of the **for** statement takes three clauses enclosed in **{**and**}** characters, as shown in this example:



Nested loops (**if**, **for**, **foreach**, and **while**) are supported.

Python example: The **Description** cell (the value of the **Command** property) of the **for** statement takes clauses enclosed in **[**and**]** characters, as shown in this example:



Nested loops (**if**, **for**, and **while**) are supported.

Adding a for loop

See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

Advanced users: Using a device parameter to set the loop count dynamically

In a more advanced case, you might use a parameter defined for the device that specifies the number of ports on the device (instead of the hard-coded number 10, as in the previous example).

1. Let's say that, for the session profile named **router**, we have defined a parameter named **port_count**, and, for this particular device, given it the value **4**.
2. Now, in the **Command** for the **for** step, we can replace the hard-coded value 10 with a **param** command that evaluates the **port_count** parameter. (The appropriate session must be specified for the steps in the **for** loop.)

```
{set port_number 0} {$port_number  [param port_count]} {incr port_number}
```

As a result, the **Command** evaluates to:

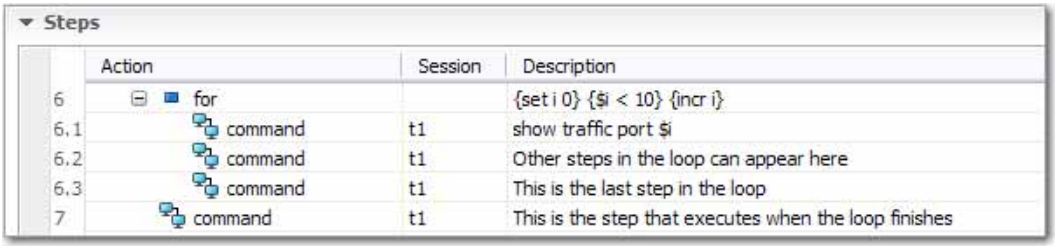
```
{set port_number 0} {$port_number<4} {incr port_number}
```

So, the loop now repeats for exactly the number of ports on the device. The test can now be used with a device with any number of ports because the port count used to control the loop dynamically takes on the port count defined for the device by the **port_count** parameter.

How for loops work

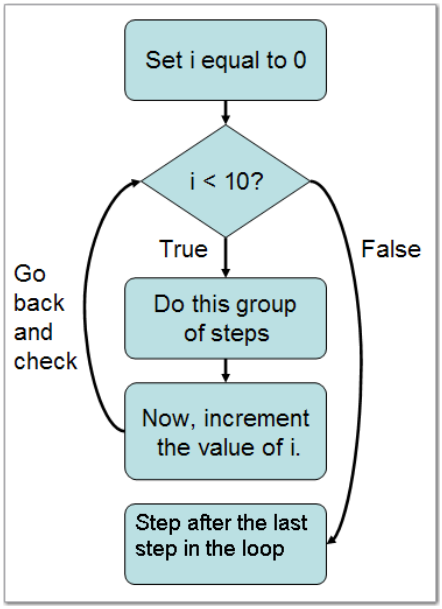
A **for** loop executes a group of steps a specified number of times. For example, repeat the steps within the loop 10 times.

Tcl example:



Python example:

Action	Session	Description
for		i in range (0, 10):
comment		Insert steps here to be executed on each loop
comment		Insert command to show i
comment		Insert other steps in the loop
comment		Last step in the loop
comment		Inset step that executes when loop finishes



Execution

The **for** loop in the example follows this logic:

1. Evaluate (execute) **set i 0** (i = 10 in Python) one time upon entering the loop. This clause initializes the value that controls the loop. The default initial value of i is 0, but you can replace 0 with any value.
2. Evaluate the expression **i < 10**. (You can replace 10 with any value greater than the initial value of i.)

   If True, then continue execution at the next step.

   If False, then exit the loop by executing the step after **for** construct.

Tip Use a field replacement to set the comparison value dynamically.

3. Evaluate **incr i** in Tcl or **i+=1** in Python after the last step in the **for** construct. The clause increments the value used to control the loop. (Use a negative value to decrement.)
4. Repeat steps 2 and 3 until **$i < 10** in Tcl or **1 < 10** in Python is False.

The foreach action: Execute a group of steps in a loop

Note Python supports **for** loop construct and not **ForEach**.

A **foreach** loop performs the steps within the loop for each value in a specified set of values.

The statement in the **Description** cell (the value of the **Command** property) of a **foreach** step follows Tcl foreach syntax and takes an even number of lists and goes through the lists two at a time. The first is a list of variables and the second is the list of values that the variables take on. A **foreach** loop is composed of all of the steps that are indented under the **foreach** clause.

Nested loops (**if**, **for**, **foreach**, and **while**) are supported.

Adding a foreach loop

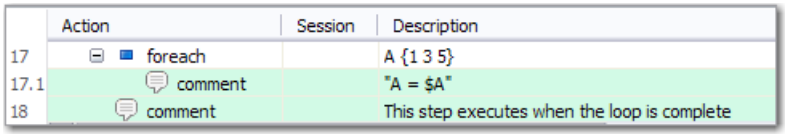
See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

How foreach loops work

A **foreach** loop performs the steps within the loop for each value in a specified set of values.

Example 1: foreach A {1 3 5}

In this example, the first list has a single member, A. The following steps result in three **comment** steps being executed with A = 1, A = 3, and A = 5:



Example 2: foreach {A B C} {a1 b1 c1 a2 b2 c2 a3 b3 c3 a4 b4 c4}

foreach also supports updating multiple variables in the same way as Tcl does. The following steps

foreach`{A B C} {a1 b1 c1 a2 b2 c2 a3 b3 c3 a4 b4}`

comment `"A = $A" "B = $B" "C = $C"`

result in four **comment** steps being executed. Notice that the second list “runs out” in the middle on the last round, so C will be equal to an empty string on the last round.

Example 3: foreach A {1 3 5} B {2 4 6 8 10}

The loop will execute for the number of iterations based on the largest value pair specified The example loop executes for 5 iterations where A,B equals:

`1,2 – 3,4 – 5,6 – <no A value>,8 – <no A value>,10`

Using a parameter to supply values

You can use a parameter with multiple values to supply a set of values. In this example, the parameter is named **A**. Use the following syntax:

```
foreach A {1 3 5}
```

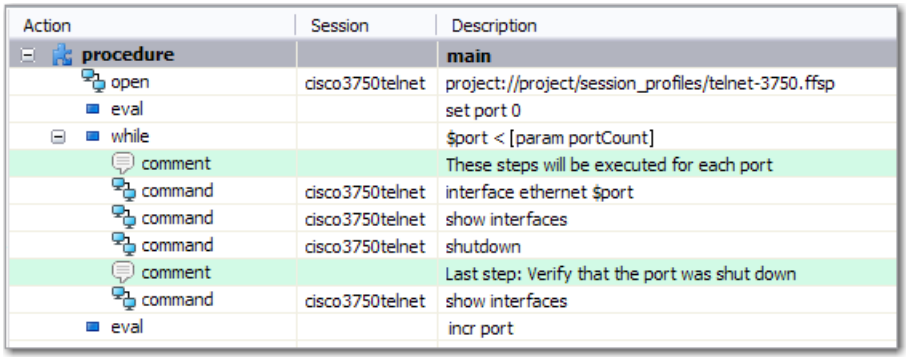
For information on defining a parameter that has multiple values, see “Defining a parameter” on page 735.

While loops

The while action: Repeat the steps in a 'while' loop

A **while** loop repeats a group of steps until a specified condition is no longer true. The **while** loop consists of a group of steps and a condition. The condition is first evaluated. If the condition is true, then the steps are executed. This repeats until the condition becomes false.

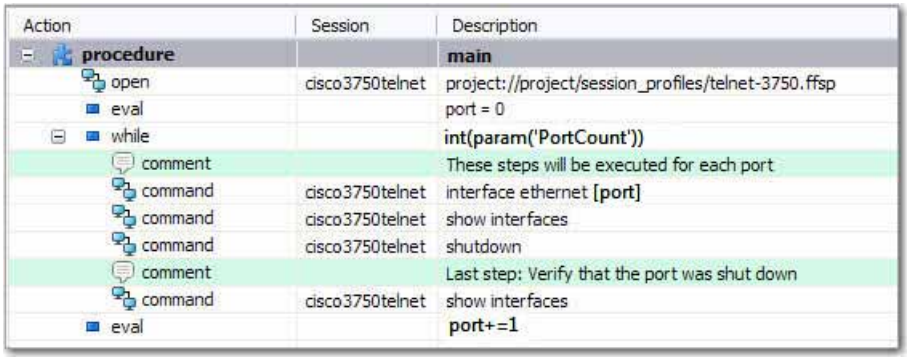
Tcl example, **while $port < param portCount** means: While the value of the **port** variable is less than the number of ports on the card (the value of the portCount parameter), repeat the loop. As soon as the **port** number is greater than or equal to **portCount**, leave the loop and continue executing at the first step after the **while** loop.



Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		set port 0
while		$port < [param portCount]
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet $port
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		incr port

Nested loops (**if**, **for**, **foreach**, and **while**) are supported.

Python example, **while port < [param('PortCount')]** means: While the value of the **port** variable is less than the number of ports on the card (the value of the PortCount parameter), repeat the loop. As soon as the **port** number is greater than or equal to **PortCount**, leave the loop and continue executing at the first step after the **while** loop.



Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		port = 0
while		**int(param('PortCount'))**
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet **[port]**
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		**port+=1**

Nested loops (**if**, **for**, and **while**) are supported.

Adding a while loop

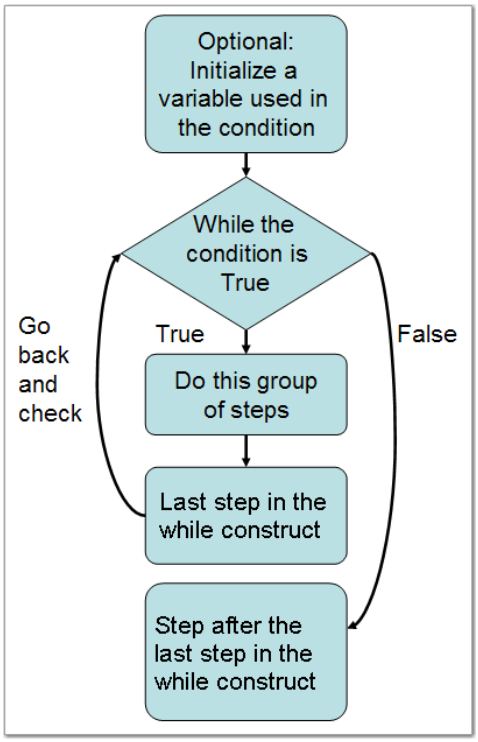
See "Inserting for, foreach, if, switch, and while constructs into a test case" on page 320.

How While loops work

A **while** loop repeats a group of steps until a specified condition is no longer true. The **while** loop consists of a group of steps and a condition. The condition is first evaluated. If the condition is true, then the steps are executed. This repeats until the condition becomes false.

For example, while the value of the **port** variable is less than the number of ports on the card (the value of the portCount/PortCount parameter), repeat the loop. As soon as the **port** number is greater than or equal to **portCount**, leave the loop and continue executing at the first step after the **while** loop.

Notice that the **while** logic checks the condition before the steps within the loop are executed, so you'll typically initialize the variable that is tested in the condition.



Execution

The following process continues until the assertion is false, or the step's timeout (if any) fires, or until execution times out or is canceled:

1 Evaluate the Tcl expression that appears in the **Command** field. (You can change the example **$i < 5** (**i < 5** in Python) expression to meet your need.)

Tip Use a field replacement to set the comparison value dynamically. The example uses a field replacement that evaluates a parameter named **portCount**.

2 If the expression is True, then execute all steps in the **while** construct.

Note Typically, one of the steps changes a value that affects the condition — this is what enables the truth of the condition to change. In the example, the EXEC **eval incr port** step changes the value.

3 Return to the **while** step and evaluate the expression.

If expression is True, then repeat execution.

If the expression is False, then skip to the step after the **while** construct.

Example Tcl

Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		set port 0
while		$port < [param portCount]
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet $port
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		incr port

The first **eval** command initializes the **port** variable.

In the **while** command, the expression **$port<[param portCount]** (**port < [param('PortCount')]** in Python) compares `port` to the upper limit of port count for the device to ensure that `port` is a reasonable port number for the device. (The port count for the device is determined by a command field replacement that evaluates the **portCount** parameter [whose value is set in the device or testbed definition].)

The **interface ethernet $port** action performs the **interface ethernet** command for the current **port** number.

The last **eval** action implements a intrepreter (Tcl or Python) command field replacement. Tcl interprets **incr port** to increment the **port** variable and return its new value. Python and interprets **port+=1** to increment the **port** variable and return its new value. This action fetches the value for the **show interfaces** command, and also has the side-effect of providing the new value for the **while** command to test.

When the CLI command analysis ends, the **while** command is again evaluated. Once the value of **port** exceeds the **portCount** for the device, the **while** command ends and execution continues at the step after the **while** construct.

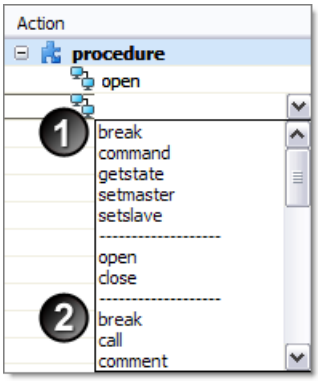
Example Python

Action	Session	Description
procedure		**main**
open	cisco3750telnet	project://project/session_profiles/telnet-3750.ffsp
eval		port = 0
while		**int(param('PortCount'))**
comment		These steps will be executed for each port
command	cisco3750telnet	interface ethernet **[port]**
command	cisco3750telnet	show interfaces
command	cisco3750telnet	shutdown
comment		Last step: Verify that the port was shut down
command	cisco3750telnet	show interfaces
eval		**port+=1**

Loop control actions

The ‘break’ action: Break out of a loop

There are two distinct kinds of b**reak** action:



- **Break CLI session execution** (marked #1 in the example): The **break** that appears in the first group of actions sends the break character for CLI sessions (typically Ctrl+C). See “The break action: Send the break character” on page 244.
- **Break out of a loop** (marked #2 in the example): The **break** that appears in the list of EXEC actions breaks out of a **for**, **foreach (in Tcl)**, or **while** loop. Use this **break** action to stop executing a loop and continue executing at the step after the loop.
  - The **Start this step in a new thread and proceed to the next step** (asynchronous execution) property on a **break** step is ignored.
  - Steps nested inside a **break** step are never used.

Example Tcl: Breaking out of a for loop

Action	Session	Description
for		{set i 0} {$i < 20} {incr i}
command	t1	show traffic
if		pktsSent != pktsRcvd
break		
command	t1	These are the steps in the loop
command	t1	This is the step that executes upon a break

In the example, the loop runs for 20 iterations. The moment that **pktsSent** is not equal to **pktsRecd**, the **break** step terminates the loop — execution then moves to the next step after the loop steps.

Example Python: Breaking out of a for loop

Action	Session	Description
for		i in range (0, 10):
command	t1	show traffic
if		pktsSent == pktsRcvd:
break		
command	t1	These are the steps in the command
command	t1	This is the step that executes upon a break

In the example, the loop runs for 10 iterations. The moment that **pktsSent** is not equal to **pktsRecd**, the **break** step terminates the loop — execution then moves to the next step after the loop steps.

The ‘continue’ action: Interrupt a loop iteration

The **continue** action causes the current script to be aborted out to the innermost containing **for**, **foreach**, or **while** loop command. The loop then continues with the next iteration of the loop.

Note Python does not use **foreach** construct in a loop.

Use the **continue** action when you want to execute particular steps for some iterations of the loop, but not for other iterations.

- The **Start this step in a new thread and proceed to the next step** property (asynchronous execution) on a **continue** step is ignored.
- Steps nested inside a **continue** step are never used.

If / then / else logic

The ‘if’ action: Element of an if/then or if-elif-else construct See page 329.

The ‘then’ action: Element of an if/then construct (Tcl) See page 331.

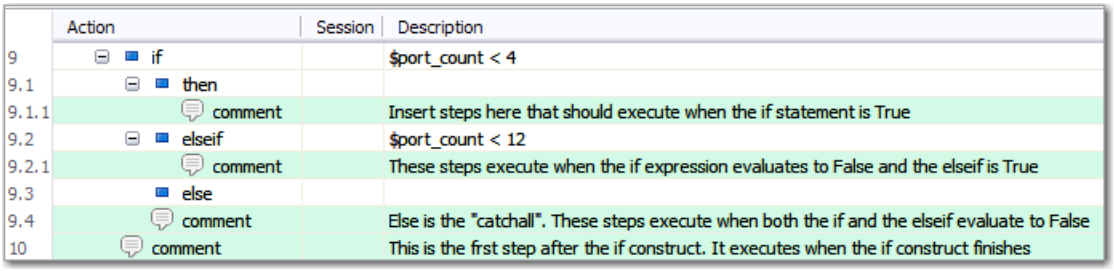
The ‘else’ action: Element of an if/then or if-else-elif construct See page 332.

The ‘elseif’/‘elif’ action: Element of an if/then or if-elif-else construct See page 333.

The ‘if’ action: Element of an if/then or if-elif-else construct

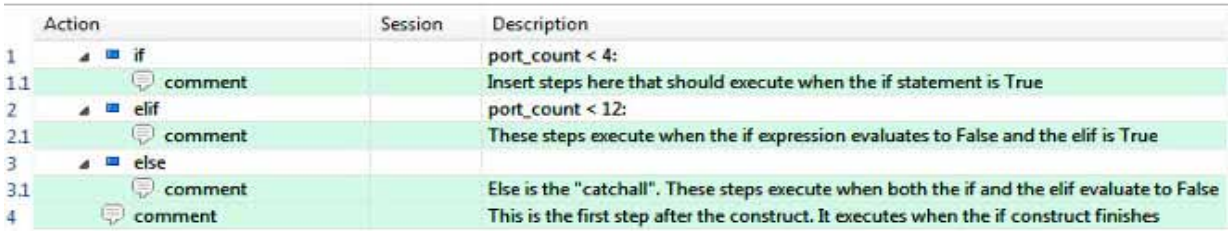
An **if** step evaluates the expression that appears in the **Description** cell (the value of the **Command** property).

Tcl example:



	Action	Session	Description
9	if		$port_count < 4
9.1	then		
9.1.1	comment		Insert steps here that should execute when the if statement is True
9.2	elseif		$port_count < 12
9.2.1	comment		These steps execute when the if expression evaluates to False and the elseif is True
9.3	else		
9.4	comment		Else is the "catchall". These steps execute when both the if and the elseif evaluate to False
10	comment		This is the frst step after the if construct. It executes when the if construct finishes

Python example:



	Action	Session	Description
1	if		port_count < 4:
1.1	comment		Insert steps here that should execute when the if statement is True
2	elif		port_count < 12:
2.1	comment		These steps execute when the if expression evaluates to False and the elif is True
3	else		
3.1	comment		Else is the "catchall". These steps execute when both the if and the elif evaluate to False
4	comment		This is the first step after the construct. It executes when the if construct finishes

If the expression is True, then continue execution at the immediately following step.

If the expression is False, then continue execution at the associated **elseif** action (**elif** action for Python). If there is no **elseif/elif** action, or if the **elseif/elif** action evaluates to False, then continue execution at the associated **else** action.

When the steps indented under the **else** are complete, then continue execution at the immediately following step (that is, after the last step in the **if** construct).

Adding an if-then-else-elseif or if-elif-else construct

See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

Nested loops (**if**, **for**, **foreach**, and **while**) are supported.

Note You can use field substitutions in the **if** clause.

else and **elseif** (**elif** for Python) steps in an **if** construct use the **Start this step in a new thread and proceed to the next step** (asynchronous execution) property of the **if** step. The asynchronous execution setting of **else** and **elseif**/**elif** steps are ignored.

You can specify multi-line expressions by clicking **Advanced** for the **command** property.

Do not define analysis rules for **else** steps.

You can nest **if-then-else** or **if-elif-else** constructs.

If an **if** step is skipped, then the entire **if** construct is skipped. However, you can skip individual **else** and **elseif/elif** constructs within the **if** construct.

How if constructs are structured

An **if** construct is composed of all of the steps that are indented under the **if**, **then**, **elseif**, and **else** clauses in Tcl and **if**, **elif**, and **else** clauses in Python. The **Command** for **if** is an expression. (You can change the example expression to meet your need.)

The optional **elseif** and **else** actions define the beginning of the steps to execute in the case that the **if** expression evaluates to False. The **elseif** step must appear before the **else** step.

Indentation of steps defines the end of an **if** construct.

Structure of an ‘if’

A legal **if** construct is made up of a contiguous sequence of:

One **if** step

Followed by one **then** step (or nested steps in Tcl)

Followed by zero or more **elseif**/**elif** steps (or nested steps)

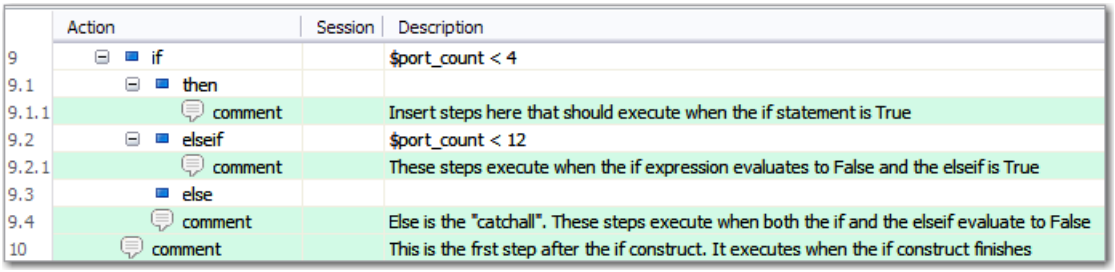
Followed by zero or one **else** step (or nested steps)

Comments are allowed at any point. Any other sequence is illegal. No other type of step can be interleaved in the sequence.

The ‘then’ action: Element of an if/then construct (Tcl)

Note Python uses the **if-else-elif** construct. See “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329.

A **then** construct can appear only as the first construct within an **if** construct. The steps that are indented under the **then** step are executed only when the **if** condition (in the example, **$port_count < 4**) evaluates to True.



	Action	Session	Description
9	if		$port_count < 4
9.1	then		
9.1.1	comment		Insert steps here that should execute when the if statement is True
9.2	elseif		$port_count < 12
9.2.1	comment		These steps execute when the if expression evaluates to False and the elseif is True
9.3	else		
9.4	comment		Else is the "catchall". These steps execute when both the if and the elseif evaluate to False
10	comment		This is the frst step after the if construct. It executes when the if construct finishes

Structure

A legal **if** construct is made up of a contiguous sequence of:

One **if** step

Followed by one **then** step (or nested steps) “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329

Followed by zero or more **elseif** steps (or nested steps)

Followed by zero or one **else** step (or nested steps)

Comments are allowed at any point. Any other sequence is illegal. No other type of step can be interleaved in the sequence.

Adding an if / then / elseif / else construct

See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

See “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329.

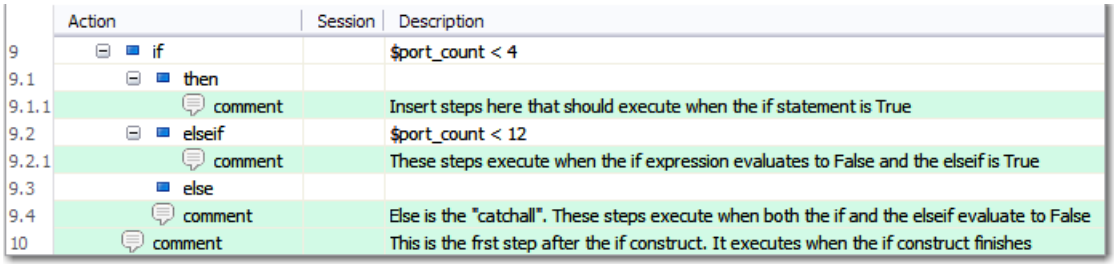
The ‘else’ action: Element of an if/then or if-else-elif construct

Note Python uses the **if-elif-else** construct and not the **if/then** construct. “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329.

An optional EXEC **else** step is a part of an **if-then-else-elseif** construct in Tcl and **if-elif-else** construct in Python.

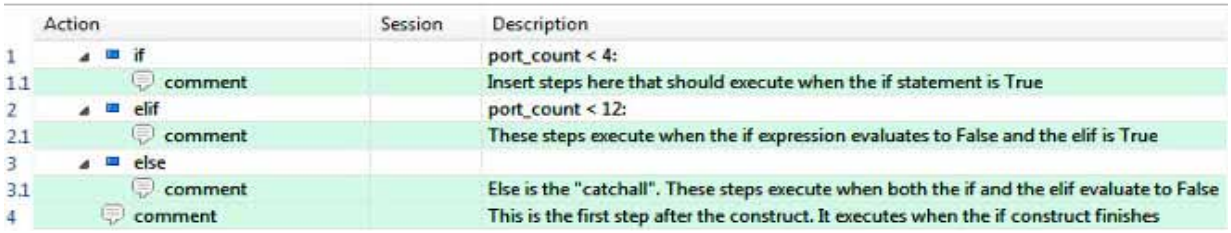
An **else** step is similar to **elseif/elif**, but it must come last in the sequence of steps associated with the **if** construct. See “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329 for full details.

If the assertion associated with the **else** is True, then its nested steps will be executed as long as no previous associated **if** or **elseif** has been actioned.



	Action	Session	Description
9	if		$port_count < 4
9.1	then		
9.1.1	comment		Insert steps here that should execute when the if statement is True
9.2	elseif		$port_count < 12
9.2.1	comment		These steps execute when the if expression evaluates to False and the elseif is True
9.3	else		
9.4	comment		Else is the "catchall". These steps execute when both the if and the elseif evaluate to False
10	comment		This is the frst step after the if construct. It executes when the if construct finishes

Python example:



	Action	Session	Description
1	if		port_count < 4:
1.1	comment		Insert steps here that should execute when the if statement is True
2	elif		port_count < 12:
2.1	comment		These steps execute when the if expression evaluates to False and the elif is True
3	else		
3.1	comment		Else is the "catchall". These steps execute when both the if and the elif evaluate to False
4	comment		This is the first step after the construct. It executes when the if construct finishes

Structure

A legal **if** construct is made up of a contiguous sequence of:

One **if** step

Followed by one **then** step (or nested steps—Tcl)

Followed by zero or more **elseif**/**elif** steps (or nested steps)

Followed by zero or one **else** step (or nested steps)

Comments are allowed at any point. Any other sequence is illegal. No other type of step can be interleaved in the sequence.

Adding an if / then / else statement

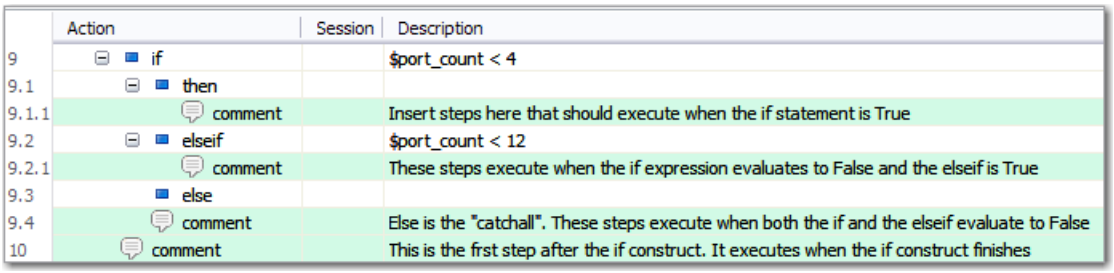
See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

The ‘elseif’/‘elif’ action: Element of an if/then or if-elif-else construct

Note Python uses the **if-elif-else** construct. “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329.

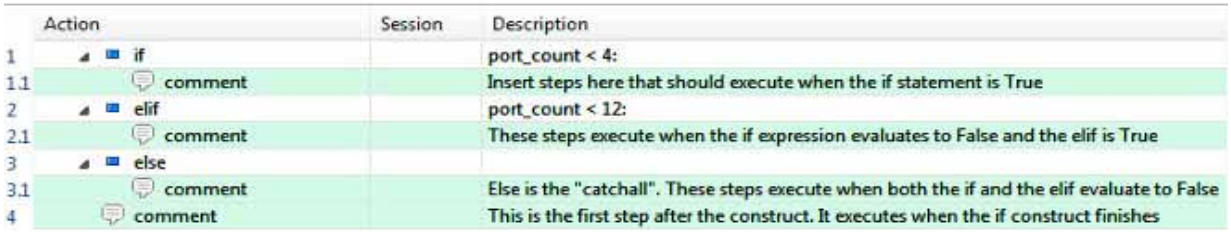
An EXEC **elseif**/**elif** step is legal only when it immediately follows an **if** statement or another **elseif**/**elif** statement. The **command** for **elseif** /**elif** contains an assertion. If no previous **if** or **elseif/elif** step that is associated with the **elseif/elif** was True and the **elseif/elif** assertion is True, then its nested steps will be executed.

If a previous **if** or **elseif**/**elif** assertion was True, then the **elseif/elif** assertion not tested.



	Action	Session	Description
9	if		$port_count < 4
9.1	then		
9.1.1	comment		Insert steps here that should execute when the if statement is True
9.2	elseif		$port_count < 12
9.2.1	comment		These steps execute when the if expression evaluates to False and the elseif is True
9.3	else		
9.4	comment		Else is the "catchall". These steps execute when both the if and the elseif evaluate to False
10	comment		This is the frst step after the if construct. It executes when the if construct finishes

Python example:



	Action	Session	Description
1	if		port_count < 4:
1.1	comment		Insert steps here that should execute when the if statement is True
2	elif		port_count < 12:
2.1	comment		These steps execute when the if expression evaluates to False and the elif is True
3	else		
3.1	comment		Else is the "catchall". These steps execute when both the if and the elif evaluate to False
4	comment		This is the first step after the construct. It executes when the if construct finishes

Structure

A legal **if** construct is made up of a contiguous sequence of:

One **if** step

Followed by one **then** step (or nested steps—Tcl)

Followed by zero or more **elseif**/**elif** steps (or nested steps)

Followed by zero or one **else** step (or nested steps)

Comments are allowed at any point. Any other sequence is illegal. No other type of step can be interleaved in the sequence.

Adding an if / then / else statement

See “Inserting for, foreach, if, switch, and while constructs into a test case” on page 320.

Substitutions in the Command property text

The command of the **if** construct is directed at the iTest interpreter. To ensure that iTest commands like [tcl] or [tclexpr] will be correctly interpreted, the text in the **Description** cell (actually, the text for the **Command** property) is interpreted as literal text.

The property that controls field replacements (command substitution) for the step is disabled and dimmed (the **For the Command field, perform command, variable, and backslash substitution** checkbox is unchecked).

As a result, the text is not processed for the following substitution types before the step is executed (substitution occurs during execution):

Command field replacements (**char**, **expr**, **param**, **query**, and **response**)

Variables

Backslash characters used to escape special characters

See “The ‘if’ action: Element of an if/then or if-elif-else construct” on page 329, “The ‘continue’ action: Interrupt a loop iteration” on page 329, and “The ‘break’ action: Break out of a loop” on page 328.

Switch logic

Note Switch logic does not apply to Python test cases

The ‘switch’ action: Control execution flow based on the value of a variable or expression (Tcl)

A **switch** construct allows the value of an expression to determine the flow of execution.

	Action	Session	Description
	procedure		**main**
1	open	s1	project://my_project/session_profiles/Telnet_myDUT.ffsp
2	eval		set i 2
3	eval		set yourCity 1
4	eval		set myCity 2
5	eval		set ourCity 3
6	switch		$i
6.1	case		$yourCity
6.1.1	eval		puts "purchasing ticket for" Mumbai
6.2	case		$myCity
6.2.1	eval		puts "purchasing ticket for" Shanghai
6.3	case		$ourCity
6.3.1	eval		puts "purchasing ticket for" London
6.4	default		
6.4.1	eval		puts "purchasing ticket for somewhere warm and sunny"

Structure and operation

A legal **switch** construct is made up of a contiguous sequence of:

One **switch** step

iTest evaluates the control clause to determine the control value. That is, iTest evaluates the expression in the **Description** cell (the value of the **Command** property). In our example, the **$i** expression (the control clause) evaluates to **2**

One or more nested **case** steps

The **Description** for each **case** step is the value of the variable or expression to compare to the control value. In our example, the values are **1**, **2**, and **3**

When a match occurs, then execution continues with the steps that are nested under the **case** step. When the last nested step finishes, then execution proceeds at the step after the **switch** construct. In our example, the steps nested under the ‘**2**’ **case** step are executed.

Zero or one nested **default** step. The **default** step must be the last step in the construct.

If the control value matches none of the **case** expressions, then execution switches to the steps that are nested under the **default** step. When the last nested step finishes, then execution proceeds at the step after the **switch** construct.

If there is no **default** step, then execution proceeds at the step after the **switch** construct

Comments are allowed at any point. Any other sequence is illegal.

About ‘switch’ constructs

- **switch** steps support iTest variables (**$i**), parameters (**[param]**), expressions (**$a + $b**), constant values, and field substitutions
- To specify a multi-line expression for a **switch** control clause, use the **Advanced** settings for the **Command** property
- You can nest **switch** constructs
- If a **switch** step is skipped, then the entire **switch** construct is skipped. You can skip any individual **case** step within a switch construct.
- The **case** and **default** steps in a **switch** construct use the **Start this step in a new thread and proceed to the next step** (asynchronous execution) property of the **switch** step. The asynchronous execution settings of **case** and **default** steps are ignored.

CHAPTER 17

iTest Commands

iTest interpreter commands

The iTest interpreter performs tasks that are useful in the iTest environment. We designed the syntax to be very much like Tcl or Python so that the commands would be easier to understand. You can use iTest interpreter commands to perform a variety of tasks; in Tcl test cases, some commands set a variable value (**set**), get a variable value (**get**), perform mathematical operations (**math.abs**). Both Tcl and Python return information about iTest (**info**), or access the response to an earlier step (**response**).

Some iTest interpreter commands have both Tcl and Python counterparts and some do not. For example, you can use **set i 0** (which uses the same syntax as Tcl) to assign the value **0** to the variable **i**. You can then use **i** as a local variable in your test case with the value **$i**.

The built-in Python interpreter starts with the following modules loaded: basic math functions, random number generation, time methods, regular expressions, file reads and writes, stdio, data structure manipulations, and JSON structure processing.

Note To learn how the iTest environment and the Tcl environment relate, see "About the iTest interpreter and the Tcl interpreters" on page 249.

Examples

In this example **eval** step, a **set** command sets the value of the **port_count** variable.

Tcl

Action	Session	Description
eval		set port_count 8

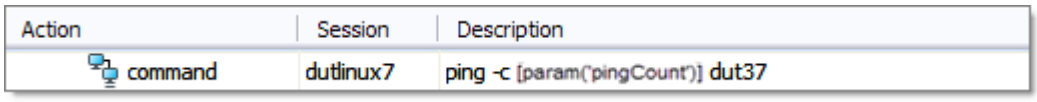
Python

Action	Session	Description
eval		port_count=8

The **param** command returns the value of a parameter. In this example, the **param** command in the **command** step is placed inside a field replacement. At runtime, before the step is interpreted, iTest substitutes the returned value for the field replacement (in this example, iTest substitutes the value of the parameter named **ping_count**). So, if the parameter had the value **9**, then the step would execute as **ping -c 9 dut37**. (Field replacements are described in Chapter 28, "Field Replacements".)

Action	Session	Description
command	dutlinux7	ping -c [param ping_count] dut37

Python



Recursion

Any command can itself include inserted commands (to any level of recursion). In this example **concat** command (that concatenates a name and a file extension to result in a proper filename), the string representing the name is substituted with the value of a parameter named **inputFile**. So, if at runtime the **inputFile** parameter has the value **myFile**, then the following construct is replaced by a filename: **myFile.fftc**

Tcl: [concat [param inputFile] .fftc]

Python:

param(param('name_of_parameter'))

gget('varName', param('my_param'))

Adding iTest interpreter commands to steps

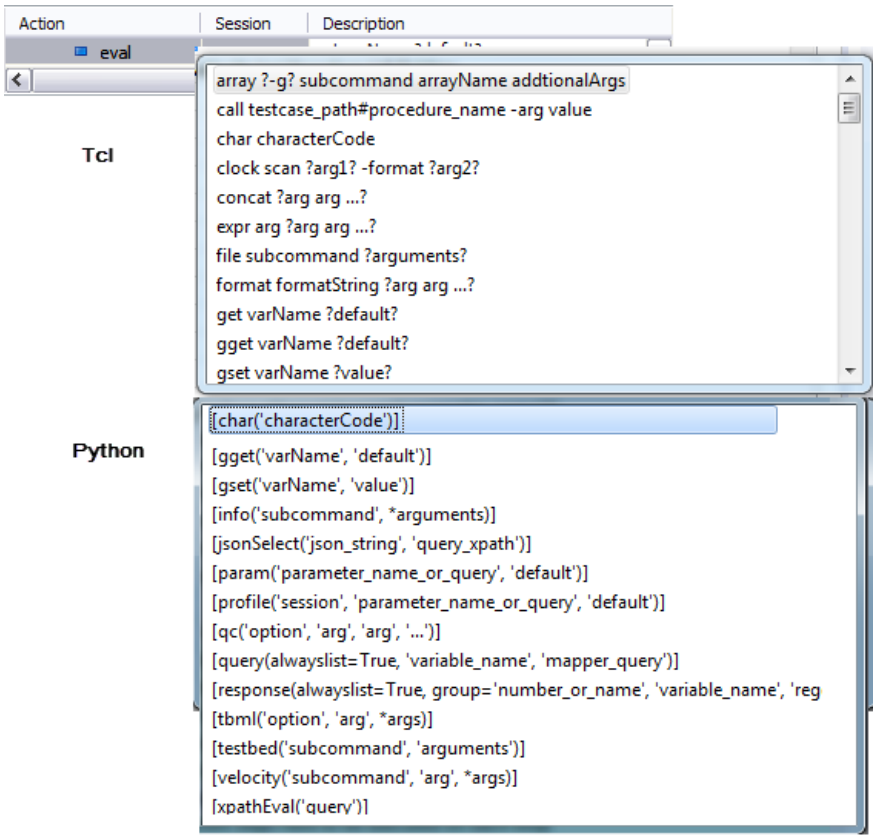
Inserting a step that executes command

1. Create a step with an **Action** of **eval**.
2. Click in the **Description** cell and then click the down-arrow to open a drop-down list of commands.

3 Select the command from the list. The command text includes “helper” text to remind you of required or optional subcommands and arguments.

Tcl: Question marks ? surrounding an item indicates that the item is optional.

Python: No preceding item to indicate that the item is optional.



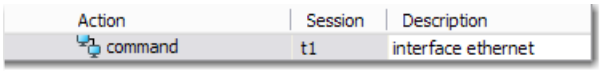
Overview: Inserting a command as a field replacement

You can insert any command as a field replacement into an existing step **Description** or into a property setting. At runtime, before the property or step is interpreted, iTest substitutes the returned value for the field replacement. The generic format for a field replacement is:

[commandName args] in Tcl **or commandName('arg')** in Python

You do not have to remember the field replacement syntax. Just right-click anywhere that you can add a field replacement and select **Insert** to insert a properly formatted field replacement with hints about argument usage.

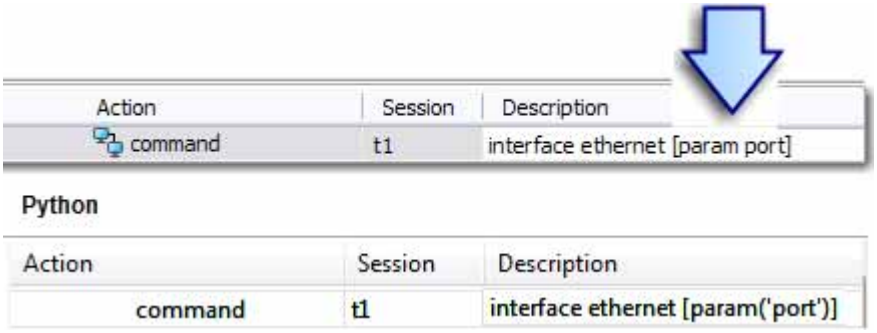
In this example, we have added the first portion of a device’s **interface ethernet** command (it is not a iTest command, rather a CLI command to the device’s management interface).



The **interface ethernet** command requires a port number as the argument (for example, **interface ethernet 9**). We want the iTest step to determine the port number dynamically at runtime from a parameter that supplies the value. We will add the port argument to the

command text as a iTest **param** command. During execution, the **param** command will be replaced by the port number.

So, we place the cursor after “ethernet”, right-click, select **Insert**, and then select **Parameter**. The **Insert Field** tool then helps us to select the particular parameter to use (we chose “**port**”) and then inserts a **param** command (within the **[** and **]** brackets of a field replacement). Now the command will execute as desired.



For more detailed instructions, see “Inserting field replacements using the Insert Field tool” on page 637. Field replacements are fully described in Chapter 28, “Field Replacements”.

Tcl interpreter local variables

- **$value** is a iTest interpreter variable that stores the data that is returned by the extractor. **$value** is created in the heap.
  - For the **contains** extractor (string comparisons), **$value** is either **1** (True, the string matches) or **0** (zero, False)
  - For the **regex** extractor, **$value** is the extracted value
  - For the **queries** extractor, **$value** is the result of the query

$itest_value is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_value** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_value** can be overwritten by another thread.

> **Note** To make use of **$itest_value**, you must have executed a **scriptEval** step at least once in the test case.

- **$index** is a iTest interpreter variable. When the extractor extracts multiple items and the processor is invoked for each item, then **$index** holds the index of each value. For example, use a value's index to chart each extracted value on a separate line or series.

$itest_index is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_index** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_index** can be overwritten by another thread.

Note To make use of **$itest_index**, you must have executed a **scriptEval** step at least once in the test case.

iTest Tcl interpreter commands

About the iTest interpreter and the Tcl interpreters

- ### Actions that operate in the iTest environment: eval, set, get

  The iTest interpreter performs tasks that are useful in the iTest environment. We designed the syntax to be very much like Tcl so that the commands would be easier to understand. You can use iTest interpreter commands to perform a variety of tasks; some commands set a variable value (**set**), get a variable value (**get**), perform mathematical operations (**math.abs**), return information about iTest (**info**), or access the response to an earlier step (**response**).

  Some iTest interpreter commands have Tcl counterparts and some do not. For example, you can use **set i 0** (which uses the same syntax as Tcl) to assign the value **0** to the variable **i**. You can then use **i** as a local variable in your test case with the value **$i**.

  You can use the iTest **eval** action to evaluate the iTest commands (actually, statements) that are specified in the **Description** cell.

The iTest interpreter and the Tcl interpreter

- ### Using special iTest actions to access the Tcl environment: scriptEval, scriptSet, and scriptGet

  The **scriptEval** action uses the Tcl interpreter. If you perform **set i 0**, then variable **i** is set only in the Tcl environment — the iTest environment does not know about the variable.

  Note Even though the variable that we just set is named **i**, it is a variable in the Tcl environment—a completely different and independent variable from the variable **i** that we earlier set in the iTest environment.

  You can continue to use **scriptEval** to, for example, source your own Tcl script that makes use of **$i**. The benefit of having **scriptEval** is that it supports all Tcl operations.

  To pass variables back and forth between the iTest interpreter and the Tcl interpreter, use **scriptSet** and **scriptGet**.

  Note scriptEval, scriptSet, and scritGet are not application in Python

- ### You can think of it as the “iTest world” and “Tcl world”:

  - When you use the **scriptEval** action, you are operating in the Tcl world.
  - When you use the **eval** action, you are operating in the iTest world.
  - You use the **scriptSet** and **scriptGet** actions to peek and poke variables between the Tcl world and the iTest world.

About Tcl commands

The table lists all iTest interpreter commands alphabetically. Some commands are described in greater detail in separate sections that group commands by function.

Some commands operate in the same way as their Tcl counterpart, some do not, as described here. You will find a Tcl tutorial at http://www.tcl.tk/man/

The "**Right-click shortcuts**" identified in the table are described in detail in "Editing test case steps: Basic tools" on page 149.

Tcl Syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
?x?	Keywords or arguments that appear within question marks are optional.
{**x** | **y** | **z**}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[**x** {**y** | **z**}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

array ?**-g**? *subcommand arrayName ?arg arg ...?*	Performs an operation on the existing array variable specified by ***arrayName***. The iTest **array** command is compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/ The optional **-g** argument indicates a global array. **Subcommands** The following subcommands are supported: **[array ?-g? compare** *array1 array2***]** Returns -1 if *array1* is less than *array2* Returns 0 if *array1* is equal to *array2* Returns 1 if *array1* is bigger than *array2* Arrays are compared recursively. **[array ?-g? exists** *arrayName***]** Returns 1 if *arrayName* is an array variable, 0 if there is no variable by that name or if it is a scalar variable. **[array ?-g? get** *arrayName* ?*pattern*?**]** In the optional *pattern*, only the ***** and **?** wildcard characters are supported. The **[chars]** and **\x** options are not supported. **[array ?-g? names** *arrayName* ?*pattern*?**]** Returns a list containing the names of all of the elements in the array that match *pattern*. The syntax differs from Tcl syntax: Tcl: **array names** *arrayName* ?*mode*? ?*pattern*? iTest: **array names** *arrayName* ?*pattern*? (*mode* always defaults to glob) In the optional *pattern*, only the ***** and **?** wildcard characters are supported. The **[chars]** and **\x** options are not supported. **[array ?-g? set** *arrayName list***]** Sets the values of one or more elements in *arrayName*. **[array ?-g? size** *arrayName***]** Returns a decimal string giving the number of elements in the array. If *arrayName* isn't the name of an array then 0 is returned. **[array ?-g? unset** *arrayName* ?*pattern*?**]** Unsets all of the elements in the array that match *pattern*. In the optional *pattern*, only the ***** and **?** wildcard characters are supported. The **[chars]** and **\x** options are not supported.
char *characterCode*	Inserts a non-printing character (for example, tab, Ctrl-C, Esc, or Delete) into a command or property. See "char command: Inserting non-printing characters" on page 371 for details. **Right-click shortcut:** Insert > Non-Printing Character

clock *option* ?*arg arg ...*?	• Return the time between one iTest step and another. • Return elapsed time by comparing two timestamps. For example timestamps taken from a log file (the format may vary) • Add or subtract from a time. For example, given a starting time, if I add 2 hours (or minutes, or days, and so on) what is the new time? The epoch (starting date and time) is 12/31/1969 16:00:00 **clock clicks -milliseconds** Returns the number of milliseconds elapsed (High resolution timer not based on any epoch) **clock** format *clockValue* ?**-format** *string*? ?**-gmt** boolean? Formats *clockValue* with the specified format string • **format**: Specifies the format of the string to be printed • **gmt**: An optional boolean variable that specifies whether or not the time is GMT **clock seconds** Returns the number of seconds elapsed since the epoch **clock milliseconds** Returns the number of milliseconds elapsed since the epoch **clock microseconds** Returns the number of microseconds elapsed since the epoch **clock scan** *dateString* ?**-base** clockVal? ?**-gmt** boolean? Scans *dateString* and converts it to the number of seconds since the epoch • **base**: The beginning date to start the clock (Default is the epoch date) • **gmt: An** optional boolean variable that specifies whether or not the time is GMT For further details, see: http://www.tcl.tk/man/
concat ?*arg arg ...*?	Concatenates all of the string representations of the arguments into a single string with whitespace between argument strings. The **concat** command joins each of its arguments together with spaces after trimming leading and trailing white-space from each of them. If all the arguments are lists, this has the same effect as concatenating them into a single list. It permits any number of arguments; if no arguments are supplied, the result is an empty string. The iTest **concat** command is compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/ **Right-click shortcut:** Insert > Special Actions > Concatenate Strings

expr *arg* ?arg arg ...?	Evaluates the specified expression into a command or property. The result of the query is converted into a string and inserted in place of this command. Examples **[expr 5 + 5]**: replaced by **10** **[expr $i + 1]**: **$i** is first substituted. If **i** has value **10**, then the result of this command is **11**. See "expr command: Evaluating expressions" on page 372 **Right-click shortcut:** Insert > Special Actions > Expression
File and directory management commands The **file** commands enable you to manage files and directories, whether in the workspace or elsewhere on the file system. Full descriptions appear in "Commands for managing files and directories" on page 363.	
format *formatString* ?*arg arg ...*?	Generates a formatted string in a fashion similar to the ANSI C **sprintf** procedure. *formatString* indicates how to format the result, using **%** conversion specifiers as in **sprint**. The additional arguments, if any, provide values to be substituted into the result. The return value from **format** is the formatted string. The **format** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
get *varName ?defaultValue?*	Returns the value of the specified local variable. If the specified variable is not found, then the command returns the default value if specified. Alternative syntax for returning the value: **${***varName***}.** The expressions **$i** and **[get i]** are identical in operation; they both return the value of the variable **i**. **Right-click shortcut:** Insert > Local Variable > Get using command See the description for **Local variable** in "Items in the Insert menu" on page 159.
gget *varName ?defaultValue?*	Returns the value of the specified global variable. Alternative syntax for returning the value: **${/data/***varName***}** The expressions **$/data/i** and **[gget i]** are identical in operation; they both return the value of the variable **i**. If the specified variable is not found, then the command returns the default value if specified. See the description for **Global variable** in "Items in the Insert menu" on page 159. **Right-click shortcut:** Insert > Global Variable > Get

gset *varName ?defaultValue?*	Sets the value of the specified global variable. Arrays: Use set a(1,2) foo syntax. (To create an array with one element, use the array command.) **Lists: gset a {1,2 foo}** creates a list with two elements, the first element is "1,2" and the second is "foo". See the description for **Global variable** in "Items in the Insert menu" on page 159.
gunset *varName*	Removes one global variable. **Limitation**: No additional arguments are allowed. The command is otherwise compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/
incr *varName*	Increments the specified local or global variable. **Right-click shortcuts:** Insert > Special Actions > Increment Local Variable Insert > Special Actions > Increment Global Variable
info *subcommand ?arguments?*	The info group of commands return information about execution, about values, about the current instance of iTest, and about the user and local computer. You can use an **eval** action with an **info** command or can use the command as a field replacement. See "Commands for returning information: info" on page 358.
join *list* ?*joinString*?	Creates a string by joining together list elements. The *list* argument must be a valid Tcl list. The command returns the string formed by joining all of the elements of *list* together with *joinString* separating each adjacent pair of elements. The *joinString* argument defaults to a space character. The **join** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
The **list** commands all start with the letter **l**.	
lappend *varName ?value value ...?*	Treats the *varName* variable as a list and appends each of the *value* arguments to the list as a separate element, with spaces between elements. The **lappend** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
lassign *list varName* ?*varName ...*?	Assigns list elements to variables. The command treats the value *list* as a list and assigns successive elements from that list to the *varName* variables in order. If there are more variable names than list elements, the remaining variables are set to the empty string. If there are more list elements than variables, a list of unassigned elements is returned. The **lassign** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/

lcompare *list1 list2*	Returns -1 if **list1** is less than **list2** Returns 0 if **list1** is equal to **list2** Returns 1 if **list1** is bigger than **list2** Lists are compared recursively.
lindex *list* ?*index*?	Retrieves an element from a list. The **lindex** command accepts a parameter, *list*, which it treats as a Tcl list. It also accepts zero or more indexes into the list. The indexes may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument. You can specify **end?-n?** as an index, so **lindex {a b c} end** will return **c** **lindex {a b c} end-1** will return **b** If no indexes are present, then the return value of **lindex** is the value of the *list* parameter. The **lindex** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
linsert *list index element* ?*element element ...*?	Inserts elements into a list. The command produces a new list from *list* by inserting all of the element arguments just before the *index*'th element of list. Each element argument will become a separate element of the new list. If *index* is less than or equal to zero, then the new elements are inserted at the beginning of the list. The interpretation of the *index* value is the same as for the command string index, supporting simple index arithmetic and indexes relative to the end of the list. The **linsert** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
list ?*arg ... arg*?	Returns a list comprised of all the args, or an empty string if no args are specified. Braces and backslashes get added as necessary, so that the **lindex** command may be used on the result to re-extract the original arguments, and also so that **eval** may be used to execute the resulting list, with `arg1` comprising the command's name and the other args comprising its arguments. **list** produces slightly different results than **concat**: **concat** removes one level of grouping before forming the list, while **list** works directly from the original arguments. The **list** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
llength *list*	Counts the number of elements in a list. The command `treats` *list* as a list and returns a decimal string giving the number of elements in it. The **llength** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/

lrange *list first last*	Returns one or more adjacent elements from a list. *list* must be a valid Tcl list. The command returns a new list consisting of elements *first* through *last*, inclusive. The index values *first* and *last* are interpreted the same as index values for the command string index, supporting simple index arithmetic and indexes relative to the end of the list. The **lrange** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
lrepeat *number element1 ?element2 element3 ...?*	Builds a list by repeating elements. The command creates a list of size *number * number of elements* by repeating *number* times the sequence of elements *element1 element2 ....* *number* must be a positive integer, *elementn* can be any Tcl value. Note that **lrepeat 1 arg ...** is identical to **list arg ...**, though the *arg* is required with **lrepeat**. The **lrepeat** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
lreplace *list first last ?element element ...?*	Replaces elements in a list with new elements. **lreplace** returns a new list formed by replacing one or more elements of list with the *element* arguments. *first* and *last* are index values specifying the first and last elements of the range to replace. The index values *first* and *last* are interpreted the same as index values for the command string index, supporting simple index arithmetic and indexes relative to the end of the list. 0 refers to the first element of the list, and end refers to the last element of the list. If list is empty, then *first* and *last* are ignored. If *first* is less than zero, it is considered to refer to before the first element of the list. For non-empty lists, the element indicated by *first* must exist or *first* must indicate before the start of the list. If *last* is less than first, then any specified elements will be inserted into the list at the point specified by first with no elements being deleted. The *element* arguments specify zero or more new arguments to be added to the list in place of those that were deleted. Each *element* argument will become a separate element of the list. If no *element* arguments are specified, then the elements between *first* and *last* are simply deleted. If *list* is empty, any *element* arguments are added to the end of the list. The **lreplace** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
lreverse *list*	Reverses the order of a list. The command returns a list that has the same elements as the input *list*, except with the elements in the reverse order.

lsearch ?**-inline**? *list pattern*	Determines whether a list contains a specified element. If found, returns the zero-based index of the matching item. If you use the optional **-inline** switch, then returns the matching item. If not found, returns -1. *pattern* supports regex matching and exact matching. Wildcard (glob-style) matching supports using only the * and ? wildcard characters and does not support **[chars]** and **\x**. The **lsearch** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/
lset *varName ?index...? newValue*	Accepts a parameter, *varName*, which it interprets as the name of a variable containing a Tcl list. The command also accepts zero or more indices into the list. The indices may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument. Finally, the command accepts a new value for an element of *varName*. For additional detail, see the fuller description at: http://www.tcl.tk/man/
lsort ?*options*? *list*	Sorts the elements of *list*, returning a new list in sorted order. The **lsort** command performs O(n log n) sort. ASCII sorting is used by default, with the result returned in increasing order. However, any of several options may be specified to control the sorting process. **Limitation** The **-command** option is not supported. Aside from the stated limitation, the **lsort** command is compatible with its Tcl counterpart, as more fully described at: http://www.tcl.tk/man/

math.abs *arg* **math.acos** *arg* math.asin *arg* math.atan *arg* math.avg ?*arg ... arg* math.c?eil *arg* math.cos *arg* math.cosh *arg* math.double *arg* math.exp *arg* math.floor *arg* math.fmod x y math.hypot x y math.int *arg* math.log *arg* math.log10 *arg* math.max *arg ... arg* math.min *arg ... arg* math.pow x y math.rand math.round *arg* math.sin *arg* math.sinh *arg* math.sqrt *arg* math.srand *arg* math.tan *arg* **math.tan**h *arg*	Each of the iTest math function commands is compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/tcl8.4/TclCmd/expr.htm#M20 **Example 1** **expr {[math.sin $i]/2}**: In this example, notice that the syntax for the iTest **expr** command differs from the Tcl syntax: **expr {sin($i)/2}** **Example 2:** **Converting Hex value to Decimal** You have **0x2e9** in a iTest variable called **i** **eval set i 0x2e9** To convert the hex string to its integer value, use math.int: **eval set j [math.int $i]** **Right-click shortcut:** Insert > Math Function > function_name **Note:** iTest displays an error ("Cannot convert to number") if the mathematical expression to evaluate is too big. This is a limitation of the TCL interpreter. Workaround: use variables to save the whole expression step by step. **Example:** set a 29174813892342 set b [math.wide [math.abs [expr 1231-$a]]] In this case b cannot be evaluated. The expression is too large. **Workaround:** Use another variable to save the result of a piece of the math expression: set c [expr 1231 - $a] set b [math.wide [math.abs $c]]
param *paramNameOrQuery ?defaultValue*?	Inserts the value of a parameter into a test case step or property. For example, the **[param ping_count]** field replacement text is replaced at runtime by the value of the **ping_count** parameter. If you specify a *defaultValue*, then the value is used if the **param** command does not return a value See "param command: Returning parameter values" on page 372. **Right-click shortcut:** Insert > Parameter

parray ?**-g**? *arrayName* ?*pattern*?	Returns an array's keys and values. In the optional *pattern*, only the ***** and **?** wildcard characters are supported. The **[chars]** and **\x** options are not supported. **Example response:** fruit(best) = peach fruit(2nd) = apple fruit(ok) = banana fruit(worst) = fly The iTest **parray** command is compatible with its Tcl counterpart as more fully described at: http://wiki.tcl.tk/man
profile *sessionID paramNameOrQuery* ?*defaultValue*?	Returns the value of a parameter that is defined in the session profile associated with a particular session. See "profile command: Accessing parameters that are defined in session profile" on page 373. **Right-click shortcut:** Insert > Parameter
puts {*string*}	Writes the *string* characters to **stdout**. **puts** normally outputs a newline character after **string**, but this feature may be suppressed by specifying the **-nonewline** switch. **Tip** **puts** returns the string into the response. Using a **get** or **gget** command substitution in the string itself is a convenient way to display a variable value. **Right-click shortcut:** Insert > Special Actions > Write string into step response
qc *option arg* ?*arg ...*?	**qc setArtifactsLocation** *URI* Sets the **artifactsFolder** property in the **QualityCenterInfo** section of the test report . When execution finishes, iTest zips the artifacts and saves the zipped file to the specified location. **qc getArtifactsLocation** [*defaultURI*] Returns the value of the **artifactsFolder** property in the **QualityCenterInfo** section of the test report. See "qc command: Getting and setting the location of the Artifacts folder" on page 953.
query ?*-alwayslist*? *varName mapperQuery*	Inserts the result of a query into a command or property. See "query command: Inserting the results of a query" on page 374. **Right-click shortcut:** Insert > Query On Stored Response
regexp ?*switches*? *regexp string* ?*matchVar*? ?*subMatchVar subMatchVar ...*?	Matches a regular expression against a string. Returns 1 if the expression matches, 0 otherwise. The command uses Java **regexps** to implement the command. The the syntax of regexp patterns is described at: http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html The iTest **regexp** command is compatible with its Tcl counterpart as more fully described at http://www.tcl.tk/man/

regsub ?*switches*? *regExp string subSpec* ?*varName*?	Performs substitutions based on regular expression pattern matching. The command matches the regular expression *regExp* against *string*, and either copies *string* to the variable whose name is given by *varName* or returns *string* if *varName* is not present. The the syntax of regexp patterns is described at: http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html The iTest **regsub** command is compatible with its Tcl counterpart as more fully described at http://www.tcl.tk/man/
response ?-alwayslist? ?-group *number_or_name***?** *varName* **?***regex***?**	You can store a response into a variable. Use the **response** command to access a response that had previously been stored in the specified variable. (Responses are stored by setting the **Store response in variable** property for a step. See"Storing a response into a variable (for use later in the test)" on page 140.) A response with zero values or multiple values is always stored in a list. You can specify whether to obtain a single extracted value as a scalar string or in a list. For details, see "response command: Accessing response data that is stored in a variable" on page 376 **Right-click shortcut**: Insert > Stored Response > Local / Global
scan *string format* ?*varName varName ...*?	Parses substrings from an input string in a fashion similar to the ANSI C **sscanf** procedure. The command returns a count of the number of conversions performed, or -1 if the end of the input string is reached before any conversions have been performed. *String* gives the input to be parsed and *format* indicates how to parse it, using **%** conversion specifiers as in **sscanf**. Each *varName* gives the name of a variable; when a substring is scanned from *string* that matches a conversion specifier, the substring is assigned to the corresponding variable. If no *varName* variables are specified, then **scan** works in an inline manner, returning the data that would otherwise be stored in the variables as a list. In the inline case, an empty string is returned when the end of the input string is reached before any conversions have been performed. The iTest **scan** command is compatible with its Tcl counterpart as more fully described at http://www.tcl.tk/man/
set *varName* ?*value*?	Sets a local variable to the specified value. Arrays: Use set a(1,2) foo syntax. (To create an array with one element, use the array command.) **Lists: set a {1,2 foo}** creates a list with two elements, the first element is "1,2" and the second is "foo".

split *string* ?*splitChars*?	Splits a string into a proper Tcl list. The command returns a list created by splitting *string* at each character that is in the *splitChars* argument. Each element of the result list will consist of the characters from *string* that lie between instances of the characters in *splitChars*. Empty list elements will be generated if *string* contains adjacent characters in *splitChars*, or if the first or last character of *string* is in *splitChars*. If *splitChars* is an empty string then each character of *string* becomes a separate element of the result list. *SplitChars* defaults to the standard white-space characters. The iTest **split** command is compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/
string *option arg* ?*arg*?	Manipulates strings by performing the string operation specified by *option*. The iTest **string** command is compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/ **Limitations:** [**string match** *args*] supports only ***** and **?** glob pattern sequences and does not support **[chars]** and **\x**. [**string replace**] does not accept Tcl's **end**. [**string trim** $attrs ,] is not supported. **string concat** *arg arg …* In addition to the standard **string** options, the iTest interpreter supports the [**string concat** *arg arg* ...] command which concatenates the string representations of all of the arguments into a single string. If all of the arguments are lists, this has the same effect as concatenating them into a single list. The command permits any number of arguments; if no arguments are supplied, the result is an empty string.

subst *arg*	Performs backslash, command, and variable substitutions on the *string* argument. The substitutions are performed in exactly the same way as for Tcl commands. As a result, the *string* argument is actually substituted twice; once by the Tcl parser in the usual fashion for Tcl commands, and again by the **subst** command. **Limitations:** No additional arguments are allowed. The command is otherwise compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man// If an error occurs during substitution, then **subst** will return the error. If a break exception occurs during command or variable substitution, the result of the whole substitution will be the string (as substituted) up to the start of the substitution that raised the exception. If a continue exception occurs during the evaluation of a command or variable substitution, an empty string will be substituted for that entire command or variable substitution (as long as it is well-formed Tcl.) If a return exception occurs, or any other return code is returned during command or variable substitution, then the returned value is substituted for that substitution. See the examples. In this way, all exceptional return codes are ``caught'' by **subst**. The subst command itself will either return an error or will complete successfully. **Examples** When it performs its substitutions, **subst** does not give any special treatment to double quotes or curly braces (except within command substitutions) so the script set a 44 subst {xyz {$a}} returns ``xyz {44}'', not ``xyz {$a}'' and the script When command substitution is performed, it includes any variable substitution necessary to evaluate the script.
tbml *subcommand arg ?arg?*	The **tbml** group of commands returns device and connection information from a topology. See “Commands that return information about topologies” on page 524.
tcl {*statement***}**	Calls on the execution kernel's Tcl interpreter, evaluates the argument in the **Description** cell as a statement, and then returns the result of that evaluation. STDOUT and STDERR are not used. See “tcl command: Evaluating Tcl statements in the execution kernel's Tcl interpreter” on page 377.
tclexpr {*expression***}**	Calls on the execution kernel's Tcl interpreter, evaluates the argument in the **Description** cell as an expression, and then returns the result of that evaluation. STDOUT and STDERR are not used. See “tclexpr command: Evaluating Tcl expressions in the execution kernel's Tcl interpreter” on page 378.

testbed *subcommands* ?*arg*?	**testbed** commands return information about the testbed devices used during execution. See "Commands that return information about testbeds" on page 381 for detailed descriptions. **testbed devices** Returns the list of testbed device names from the testbed used for execution. Returns an empty list if there is no testbed or the testbed is empty. **testbed** *deviceName propertyName* ?*defaultValue*? Returns the string value of the specified property for the specified device. **Right-click shortcut**: Insert > Testbed Property
unset *varName*	Removes one local variable. **Limitation**: No additional arguments are allowed. The command is otherwise compatible with its Tcl counterpart as more fully described at: http://www.tcl.tk/man/
xpatheval *xpathQuery*	Evaluates the specified XPath query as applied to the root node of the current stack frame in the heap.

iTest interpreter and Python commands

About the iTest interpreter and the Python interpreters

Actions that operate in the iTest environment: eval

The iTest interpreter performs tasks that are useful in the iTest environment. The built-in Python interpreter starts with these modules: basic math functions, random number generation, time methods, regular expressions, file reads and writes, stdio, data structure manipulations, and JSON structure processing.

In addiiton to using the iTest interpreter commands to perform a variety of tasks, you may may assign global data structures (iTest QuickCall setting a global variable), which can be read and written to by Python interpreter (a test case in Python syntax that calls the QuickCall).

Some iTest interpreter commands have Tcl/Python counterparts and some do not. For example, you can use assign **i = 0** to assign the value **0** to the variable **i**.

You can use the iTest Python **eval** action to evaluate the iTest commands (actually, statements) that are specified in the **Description** cell.

About Python commands

The table lists all iTest Python interpreter commands alphabetically. Some commands are described in greater detail in separate sections that group commands by function.

You will find a Python documentation at: https://www.python.org/doc/.

The "**Right-click shortcuts**" identified in the table are described in detail in "Editing test case steps: Basic tools" on page 149.

Python Syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed within single quotes (**' '**).
?x	Keywords or arguments that appear after a question mark are optional.
[**x** | **y** | **z**]	A choice of required keywords (represented by x, y, and z) appears in square brackets separated by vertical bars. You must select one.
[**x** (**y** | **z**)]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Important:

Using square brackets ([]) in Python syntax:

- Required in session steps and session profiles fields.
- Not required in non-session steps (eg: eval).

.

char*('CharacterCode')*	Inserts a non-printing character (for example, tab, Ctrl-C, Esc, or Delete) into a command or property. See "char command: Inserting non-printing characters" on page 371 for details. **Right-click shortcut:** For Non-printing characters, first click on the command description field, then right-click in the field and insert chars. Eg: **eval [char("Ctrl-C")]**
gget('*varName*', '*defaultValue*')	Returns the value of the specified global variable. See the description for **Global variable** in "Local and global variables" on page 138. **and** "Storing a response into a variable (for use later in the test)" on page 140. **Right-click shortcut:** Insert > Global Variable > Get
gset('*varName***', '***defaultValue***'**	Sets the value of the specified global variable. "**Lists: gset***('a', '[1,2,3]')"*creates a list with three elements. See the description for **Global variable** in "Local and global variables" on page 138. **and** "Storing a response into a variable (for use later in the test)" on page 140.
info('*subcommand***', *arguments)**	The info group of commands return information about execution, about values, about the current instance of iTest, and about the user and local computer. You can use an **eval** action with an **info** command or can use the command as a field replacement. See "Commands for returning information: info" on page 358.

param *('name', default_value)*	Inserts the value of a parameter into a test case step or property. For example, the **[param ('ping_count')]** field replacement text is replaced at runtime by the value of the **ping_count** parameter. If you specify a *defaultValue*, then the value is used if the **param** command does not return a value See "param command: Returning parameter values" on page 372. **Right-click shortcut:** Insert > Parameter
profile*('name', 'default_value')*	Returns the value of a parameter that is defined in the session profile associated with a particular session. See "profile command: Accessing parameters that are defined in session profile" on page 373. **Right-click shortcut:** Insert > Parameter
qc*('subcommand', 'value')*	**qc('setArtifactsLocation' '***URI***')** Sets the **artifactsFolder** property in the **QualityCenterInfo** section of the test report . When execution finishes, iTest zips the artifacts and saves the zipped file to the specified location. **qc('getArtifactsLocation'**, **'***defaultURI***'**) Returns the value of the **artifactsFolder** property in the **QualityCenterInfo** section of the test report. See "qc command: Getting and setting the location of the Artifacts folder" on page 953.
query*('variable_name', 'mapper_query', alwayslist=False)*	Inserts the result of a query into a command or property. See "query command: Inserting the results of a query" on page 374. **Right-click shortcut:** Insert > Query On Stored Response
response*('varName', alwaysList=True, regex='', group='number | name')*	You can store a response into a variable. Use the **response** command to access a response that had previously been stored in the specified variable. (Responses are stored by setting the **Store response in variable** property for a step. See"Storing a response into a variable (for use later in the test)" on page 140.) A response with zero values or multiple values is always stored in a list. You can specify whether to obtain a single extracted value as a scalar string or in a list. For details, see "response command: Accessing response data that is stored in a variable" on page 376 **Right-click shortcut**: Insert > Stored Response > Local / Global
tbml*('subcommand', 'arg') #*args*	The **tbml** group of commands returns device and connection information from a topology. See "Commands that return information about topologies" on page 524.
velocity*('subcommand', 'arg') #*args*	The **velocity** command returns device and connection information from a topology. See "Commands that return information from Velocity" on page 519.

testbed(*'subcommand' ,'args'*)	**testbed** commands return information about the testbed devices used during execution. See "Commands that return information about testbeds" on page 381 for detailed descriptions. **testbed devices** Returns the list of testbed device names from the testbed used for execution. Returns an empty list if there is no testbed or the testbed is empty. **testbed** *deviceName propertyName* ?*defaultValue*? Returns the string value of the specified property for the specified device. **Right-click shortcut**: Insert > Testbed Property
xpatheval(*'query'*)	Evaluates the specified XPath query as applied to the root node of the current stack frame in the heap.

info commands

Commands for returning information: info

The **info** commands return information about execution and execution values, about the current instance of iTest, and about the local computer. You can use an **eval** action with an **info** command or can use the command as a field replacement.

The subcommands are listed in roughly related functional groups — be sure to check all of the groups to ensure that you know about all subcommands.

Syntax

Tcl: info *subcommand* [*arguments*]

Python: info('*subcommand***')** *#*arguments*

‘info’ subcommands for directories, URIs, and workspaces

info homeDir ?**uri**? **info('homeDir', 'uri')**	Returns the path to the home directory of the current user. The format of the path is appropriate for the operating system (for example / or \). Use the optional **uri** argument to return the directory in the “**file:/**” URI format, used where iTest requires a URI argument. **Examples** info **homeDir** on a Windows computer might return C:\Documents and Settings\myName This form of the return string does not work with iTest items that require URIs, so you would use: **info homeDir uri** returns: file:/C:/Documents and Settings/myName
info paramFile ?**path**? **info('paramFile', 'path')**	Returns the fully-qualified URI of the current parameter file. (For example, project://my_ project/parameter_files/file_name.ffpt) If you use the optional **path** argument, then the URI is not returned. Instead, the command returns the path to the file in the format appropriate for the operating system.
info profile ?**path**? **info('profile', 'path')**	Returns the fully-qualified URI of the session profile (if any) associated with the currently executing step. (For example, project://my_ project/session_profiles/file_name.ffsp) If you use the optional **path** argument, then the URI is not returned. Instead, the command returns the path to the file in the format appropriate for the operating system.
info tempDir ?**uri**? **info('tempDir', 'uri')**	Returns the path to the temporary directory of the current user. The format of the path is appropriate for the operating system (for example / or \). Use the optional **uri** argument to return the directory in the “**file:/**” URI format, used where iTest requires a URI argument. **Examples** info **tempDir** on a Windows computer might return C:\Users\myName\Temp\ This form of the return string does not work with iTest items that require URIs, so you would use: **info tempDir uri** returns: file:/C:/Users/myName/Temp/
info testbedFile ?**path**? **info('testbedFile', 'path')**	Returns the fully-qualified URI of the current testbed (if any). (For example, project://my_ project/testbeds/file_name.fftb) If you use the optional **path** argument, then the URI is not returned. Instead, the command returns the path to the file in the format appropriate for the operating system. For additional testbed commands, see “Commands that return information about testbeds” on page 381.

info testCaseFile {**current**|**main**} ?*path*? **info('testCaseFile',** ['**current**'|'**main**'], '*path*'*)*	The **current** argument returns the fully-qualified URI of the test case that contains the currently executing step. (For example, project://my_ project/test_cases/file_name.fftc) The **main** argument returns the fully-qualified URI of the test case for which you clicked **Execute**. If you use the optional *path* argument, then the URI is not returned. Instead, the command returns the path to the file specified by either **main** or **current**. The format of the path is appropriate for the operating system.
info testCaseName {current|main} **info('testCaseName',** ['**current**'|'**main**']*)*	Get the test case file name only (no URI and no file extension). The current argument returns only the file name of the test case that contains the currently executing step. The main argument returns only the file name of the test case for which you clicked Execute.
info testCaseProject {**current**|**main**} **info('testCaseProject',** ['**current**'|'**main**']*)*	If you use the **current** argument, then the command returns the name of the project for the test case that contains the currently executing step. If you use the **main** argument, then the command returns the name of the project for the main test case (the test case for which you clicked **Execute**).
info testCaseProjectPath {**current**|**main**} **info('testCaseProjectPath',** ['**current**'|'**main**']*)*	If you use the **current** argument, then the command returns the path to the project of the test case that contains the currently executing step. The format of the path is appropriate for the operating system. If you use the **main** argument, then the command returns the path to the project of the test case for which you clicked **Execute**.
info time ?*formatString*? **info('time',** '*formatString*'*)*	Returns a the time in seconds since the current test case started. If you do not specify the optional format string, then the command uses Java's localized full timestamp format. An example *formatString* is **"yyyy-MM-dd HH:mm:ss.SSS"** (We use quotes to enclose the string because the space character appears in the string.) Format strings are based on Java's SimpleDateFormat. See http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html
info timestamp ?*formatString*? **info('timestamp',** '*formatString*'*)*	Returns a timestamp of current local time. If you do not specify the optional format string, then the command uses Java's localized full timestamp format. An example *formatString* is **"yyyy-MM-dd HH:mm:ss.SSS"** (We use quotes to enclose the string because the space character appears in the string.) Format strings are based on Java's SimpleDateFormat. See http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html
info version **info('version'***)*	Returns the full version number (for example, 3.1.0.3) of the current iTest instance

info workingDir ?**uri**? **info('workingDir', 'uri'**)	Returns the path of the iTest computer's "working" directory. The format of the path is appropriate for the operating system (for example / or \). Use the optional **uri** argument to return the directory in the "**file:/**" URI format, used where iTest requires a URI argument. **Examples** info **workingDir** on a Windows computer might return C:\Users\myName\Grinder\ This form of the return string does not work with iTest items that require URIs, so you would use: **info workingDir uri** returns: File:/C:/Users/myName/Grinder
info workspacePath **info('workspacePath')**	Returns the full path of the current workspace.

'info' subcommands for the computer and user

info env *varName* *info('env', 'varName')*	Returns the contents of the environment variable specified by *varName*
info hostIP **info('hostIP')**	Returns the IP address of the computer that is currently running iTest.
info hostName **info('hostName')**	Return the hostname of the computer that is currently running iTest.
info platform **info('platform')**	Returns the current operating system (**windows** or **linux**)
info user **info('user')**	Returns the user name of the current user.

'info' subcommands for execution, procedures, threads, parameters, and variables

info exists param *param* **info('exists', 'param', '***param***')**	Returns 1 (one) if the parameter specified by *param* exists, otherwise returns 0 (zero). Similar to the Tcl **info exists** command.
info exists {**local**|**global**} *varName* **info('exist', 'local|global', '***varName***')**	Returns 1 (one) if the local or global variable specified by *varName* exists, otherwise returns 0 (zero). Similar to the Tcl **info exists** command.

info issueCount {**ok|info|warning|error|all**} **info('issueCount', ['ok'|'info'|'warning'|'error'|'al'])**	Returns, for this moment in the current execution, the current number of execution issues of the specified severity
info procedure **info('procedure')**	Returns the name of the procedure that contains the currently executing step
info status **info('status')**	Returns the current execution **Result** of the test case being executed: **Pass**, **Fail**, **Abort**, or **Indeterminate**. If no test case is running, returns **Indeterminate**.
info step report **info('step', 'report')**	Returns the step ID (for example, 2.3.1.5) of the currently executing step from the test report currently being executed.
info step testCase **info('step', 'testCase')**	Returns the step ID (for example, 2.5.3) of the currently executing step.
info step sessionID **info('step', 'sessionID')**	Returns then **Session ID** (the value in the **Session** cell) of the currently executing step. **Example** To handle unexpected events (for example, **OnExecutionTimeout**) in a test case, you could define a **callProcedure** action for the event. Use the **info step sessionName** command to obtain the session ID for the step that timed out. The **callProcedure** action would pass the session ID for the step as an argument to the procedure. As a result, when the event occurs, the procedure can clean up all devices to a known state before the test case ends.
info testReportID **info('testReportID')**	Returns the report ID of the current test report. You can use the report ID to access the report.
info threadID **info('threadID')**	Returns the ID of the current thread
info time **info('time')**	Returns the number of seconds (floating point) since execution of the current test case started, excluding pauses. If not currently executing, returns 0.

File and directory management commands

Commands for managing files and directories

The **file** group of commands enables you to manage files and directories, whether in the workspace or elsewhere on the file system.

file copy [**-y**] *sourceURI destinationURI*	Copies files from the specified source directory or file URI to the specified destination. Creates destination directories as needed. See "file copy command: Copying files to a destination URI" on page 364.
file delete [**-r**] *URI*	Deletes files from the specified directory or file URI. See "file delete command: Delete files" on page 365.
file exists *URI*	Returns 1 (one) if the file or directory exists, otherwise returns 0 (zero). See "file exists command: Determine whether a file or folder exists" on page 365.
file isDirectory *URI*	Returns 1 (one) if the URI is a directory, otherwise returns 0 (zero). See "file isDirectory command: Determine whether a URI represents a folder name" on page 366.
file isFile *URI*	Returns 1 (one) if the URI is a filename, otherwise returns 0 (zero). See "file isFile command: Determine whether a URI represents a filename" on page 365.
file list [**-r**] [**-p**] [**-nolimit**] *URI*	Lists all files and subdirectories for the specified directory or file URI See "file list command: List the files in a URI" on page 366.
file mkdir *URI*	Creates the specified directory See "file mkdir command: Add a directory" on page 367.
file mkTempDir [**-k**] [*prefix*] [*suffix*]	Creates a temporary directory See "file mkTempDir command: Create a unique temporary directory" on page 367.
file mkTempFile [**-k**] [*prefix*] [*suffix*]	Creates a temporary file. See "file mkTempFile command: Create a unique temporary file" on page 368.
file move [**-y**] *sourceURI destinationURI*	Moves files from the specified source directory or file URI to the specified destination. Creates destination directories as needed. The **move** command can rename individual files and directories. See "file move command: Move or rename files to a destination URI" on page 368.
file pathToUri *path*	Returns the URI for the specified directory or file path See "file pathToUri command: Determining the URI of a path" on page 369
file rmdir *URI*	Deletes the specified directory. See "file rmdir command: Delete a directory" on page 369.
file uriToPath *URI*	Returns the full operating system path for the specified URI (using appropriate path syntax). See "file uriToPath command: Determining a path from a URI" on page 370.

Guidelines for using URIs in file commands

For all commands in the **file** group, the following requirements apply:

- The URI argument cannot be a native file path — it must be a URI (for example, **project://my_project/myFiles**).
- Relative URIs, file URIs, and project URIs are supported (for example, **tmp/*.txt**, **file:/c:/mypath**, **project://my_project/tmp**). Relative URIs are resolved relative to the current test case URI.
- Parent directory references ".." are supported.
- Field replacements are not supported in URIs.

file copy command: Copying files to a destination URI

Copies files from the specified source directory or file URI to the specified destination.

Syntax

file copy [**-r**] [**-y**] *sourceURI destinationURI*

Description

See "Guidelines for using URIs in file commands" on page 364.

- You can use the * wildcard character in *sourceURI* but not in *destinationURI*
- If *sourceURI* is a directory name, then *destinationURI* is interpreted as a directory name.
- If *sourceURI* is a filename, then:
  - If there is no directory with that name, *sourceURI* is interpreted as a filename
  - If there is an existing directory with that name, the new file is created (with the source filename) in the destination directory
- If multiple source files are specified (by using the * wildcard in the last segment of the URI), then the destination is interpreted as a directory.
- You can use the * wildcard to represent subdirectories.
- If needed, the destination directory and appropriate parent folders are created.

Arguments

-r — Recursive (optional). Copies files and subdirectories recursively for the specified source filename.

-y — Yes (optional). Overwrites existing files. Generates an **OnFileAlreadyExists** event.

Response

Returns the number of files copied

Related events

OnFileAlreadyExists, **OnFileCopyFailed**, **OnDirCreateFailed**

Examples

file copy -r tmp save	Copies directory 'tmp' (relative to test case URI) and its contents to a new relative directory 'save'
file copy tmp/my.log file:/c:/logfiles/new.log	Copies my.log file to new.log
file copy ../my.log [file pathToUri [info homeDir]]	Copies my.log file from parent directory to user's home directory
file copy -y project://my_project/*.log save	Copies all log files to 'save' directory relative to test case URI

file delete command: Delete files

The **file delete** command deletes files from the specified directory or file URI. If all files in a directory are deleted, the directory is ***not*** deleted. Deletes a directory when the URI refers to a directory.

Syntax

file delete [**-r**] *URI*

Description

See "Guidelines for using URIs in file commands" on page 364.

- Use the * wildcard in *URI* to represent directories or subdirectories.
- Specify multiple files by using the * wildcard in the last segment of *URI*.

Optional arguments

-r — Recursive (optional). Deletes a subdirectory recursively for the specified URI.

Response

Returns the number of files deleted

file exists command: Determine whether a file or folder exists

Returns 1 (one) if the file or directory exists, otherwise returns 0 (zero)

Syntax

file exists *URI*

See "Guidelines for using URIs in file commands" on page 364.

file isFile command: Determine whether a URI represents a filename

Returns 1 (one) if the URI exists and is a filename, otherwise returns 0 (zero)

Syntax

file isFile *URI*

See "Guidelines for using URIs in file commands" on page 364.

file isDirectory command: Determine whether a URI represents a folder name

Returns 1 (one) if the URI exists and is a directory, otherwise returns 0 (zero)

Syntax

file isDirectory *URI*

file list command: List the files in a URI

Returns the list of files in the specified URI. If the URI does not exist, returns an empty list.

Syntax

file list [**-r**] [**-p**] [**-nolimit**] *URI*

See "Guidelines for using URIs in file commands" on page 364.

- Use the * wildcard in *URI* to represent directories or subdirectories
- Specify multiple files by using the * wildcard in the last segment of *URI*
- A list of URIs is not supported

Arguments

-r — Recursive: List the contents of a directory and all sub-directories recursively.

-p — Return the list of URIs in user-readable format; each URI appears on a separate line.

-nolimit — Do not limit the length of the returned list. (For a large directory, this can be very slow and an out-of-memory error is possible.) Default is to limit the list to 5000 URIs .

Response

Returns a text list of URIs for all matching files and folders. The list is limited to 5000 files (see the **-nolimit** argument).

If the file or directory URI does not exist, returns an empty list (no error).

Notes on usage

If the URI is a directory, then all files in the directory are listed.

If the **-r** argument is used, then the contents of all subdirectories are also listed (this behavior is similar to "**ls -R** *dir* in a Linux command shell).

If the directory name is matched with a wildcard (for example, **file list temp***), then each matching directory URI is also listed along with its contents.

If the URI is for a filename then the filename is listed. For example **file list $dir/*.log** lists all log files in directory $dir and **file list *.fftc** lists all test cases in the same folder as the current test case.

A test case can iterate over the list and use **[file isFile** *URI***]** or **[file isDirectory** *URI***]** to help parse the list (provided the **-p** argument is not used).

If the list URI argument is relative, then a list of project URIs is returned (for example, p**roject://my_project/test_cases/temp/...** for relative **temp** URI).

If the URI argument is for the OS file system, then a list of file URIs is returned (for example, **file:/c:/myfiles/...**).

Examples

file list project://my_project/log/*.log	might return: project://my_project/log/file1.log project://my_project/log/file2.log
file list -r project://my_project/log	... for the same directory will, in addition, return a list of URIs for all files and sub-directories: project://my_project/log/file1.log project://my_project/log/file2.log project://my_project/log/foo/ project://my_project/log/foo_/file3.log

file mkdir command: Add a directory

Creates the specified directory. If a directory already exists at the specified location, then the **mkdir** command succeeds and has no effect.

Syntax

file mkdir *URI*

See "Guidelines for using URIs in file commands" on page 364.

Response

Returns empty result on success: ""

Error cases

An error occurs if the directory does not already exist and a new directory could not be created.

file mkTempDir command: Create a unique temporary directory

Create a unique temporary directory named **iTestTempDir_*****, where ***** is a random number, in the location set by the **java.io.tmpdir** property.

By default, the temporary directory and its contents are deleted when the current test case execution has completed (or aborted). Use the **-k** option to keep the directory.

Syntax

file mkTempDir [**-k**] [*prefix*] [*suffix*]

Arguments

-k — Optional. Keep temporary directory after test case execution has completed (or aborted).

prefix — Optional. Set the prefix for directory name. Default: **iTestTempDir_**

suffix — Optional. Set the suffix for directory name. Default: no suffix

Response

Returns the URI of the temporary directory.

Examples

set mydir [file mkTempDir] returns: [tempdirUri]/iTestTempDir_45376/

set mydir [file mkTempDir my] returns: [tempdirUri]/my45377/

set mydir [file mkTempDir my bah] returns: [tempdirUri]/my45378bah/

file mkTempFile command: Create a unique temporary file

Create a unique temporary file named **iTestTempFile_***, where ***** is a random number, in the location set by the **java.io.tmpdir** property.

By default, the temporary file is deleted when the current test case execution has completed (or aborted). Use the **-k** option to keep the file.

Syntax

file mkTempFile [**-k**] [*prefix*] [*suffix*]

Arguments

-k — Optional. Keep temporary file after test case execution has completed (or aborted).

prefix — Optional. Set the prefix for filename. Default: **iTestTempFile_**

suffix — Optional. Set the filename extension. Default: **.tmp**

Response

Returns the URI of the temporary file.

Examples

set myfile [file mkTempFile] returns: [tempdirUri]/iTestTempFile_38343.tmp

set myfile [file mkTempFile my] returns: [tempdirUri]/my38344.tmp

set myfile [file mkTempFile my .log] returns: [tempdirUri]/iTestTempFile_38345.log

file move command: Move or rename files to a destination URI

Moves or renames files from the specified source directory or file URI to the specified destination.

Syntax

file move [**-y**] *sourceURI destinationURI*

Description

See “Guidelines for using URIs in file commands” on page 364.

You can use the * wildcard character in the *sourceURI* but not in *destinationURI*

- *sourceURI* must be for an existing folder or files; an error occurs if the source does not exist.
- If *sourceURI* is a directory name, then *destinationURI* is interpreted as a directory name.
- If *sourceURI* is a filename, then:
  - If there is no directory with that name, *sourceURI* is interpreted as a filename (equivalent to rename)

- If there is an existing directory with that name, the new file is created (with the source filename) in the destination directory

- If multiple source files are specified (by using the * wildcard character in the last segment of *sourceURI)*, then the destination is interpreted as a directory.
- Use the * wildcard in *URI* to represent directories or subdirectories.
- If needed, the destination directory and appropriate parent folders are created.

Arguments

-y — Yes (optional). Overwrites existing files. Generates an **OnFileAlreadyExists** event.

Response

Returns the number of files moved.

Returned count is 0 when empty directories are moved or renamed.

Returned count is limited to 5000 to ensure reasonable performance after a large directory has been moved or renamed. This does not limit the number of files moved.

Related events

OnFileAlreadyExists, **OnFileMoveFailed**, **OnDirCreateFailed**

Examples

file copy -r tmp save	Copies directory 'tmp' (relative to test case URI) and its contents to a new relative directory 'save'
file copy tmp/my.log file:/c:/logfiles/new.log	Copies my.log file to new.log
file copy ../my.log [file pathToUri [info homeDir]]	Copies my.log file from parent directory to user's home directory
file copy -y project://my_project/*.log save	Copies all log files to 'save' directory relative to test case URI

file pathToUri command: Determining the URI of a path

Returns the file URI for the specified operating system-specific absolute path.

Syntax

file pathToUri *path*

See “Guidelines for using URIs in file commands” on page 364.

Response

Returns the file URI.

An error occurs if the path is relative.

file rmdir command: Delete a directory

Deletes the specified folder and its contents recursively. An error occurs if the folder exists but delete fails.

file rmdir *URI* is equivalent to **file delete -r** *URI* except that **rmdir** does not generate an error if the directory does not exist.

Syntax

file rmdir *URI*

See “Guidelines for using URIs in file commands” on page 364.

Response

Returns empty result on success: ""

Error cases

An error occurs if directory exists but could not be removed.

file uriToPath command: Determining a path from a URI

Returns the full operating system-specific path to the directory or file at the specified URI.

Syntax

file uriToPath *URI*

See “Guidelines for using URIs in file commands” on page 364.

The URI can be relative, project, or absolute file URI.

Commands that are commonly used in field replacements

Common field replacement commands

char command: Inserting non-printing characters

Inserts a non-printing character (for example, tab, Ctrl-C, Esc, or Delete) into a command or property. You can find documentation on the ASCII character set on the Internet.

Tip You can also type escape sequences for non-printing characters directly into any command or property text.

Inserting a char command

Place the cursor where you want to insert the field replacement text (for example, a **char** command or escape sequence). Right-click, select **Non-Printing Character**, and then select the character to insert.

Syntax

char *characterCode*

char(*'CharterCode'*)

Character codes for the char command

Character code	Tcl	Python
backspace	[char bs]	char('bs')
Ctrl -C (and all other Control characters)	[char ctrl-c] Use ctrl-a through ctrl-z and ctrl-@ through ctrl-]	char('ctrl-c') Use ctrl-a through ctrl-z and ctrl-@ through ctrl-]
Escape	[char ESC]	char('ESC')
Delete	[char del]	char('del')
Tab	[char tab]	char('tab')

Escape sequences

For any character that does not appear in the table, **/character** results in **character**. For example, **/z** returns **z**.

ASCII Hex	\x<2 hex digits> Use a backslash character followed by **x**, followed by one or two hex digits. For example, Null (ASCII zero) is **x0**
Octal	\<3 octal digits> Use a backslash character followed by one, two, or three octal digits. Non-octal digits are ignored.
Unicode	\u<4 Unicode digits> Use a backslash character followed by **u**, followed by one, two, three, or four hex digits.
Backslash ('\')	\B
Backspace	\b

Bell	\a
Carriage return	\r
Escape	\e
Form feed	\f
Newline (linefeed)	\n
Tab	\t
Vertical tab	\v

expr command: Evaluating expressions

Note Applies to **Tcl** test cases

Evaluates the specified expression. The result is converted into a string and inserted in place of the command.

Syntax

expr *mathematicalExpression*

Examples

[expr 5 + 5]: replaced by **10**

[expr $i + 1]: **$i** is first substituted with the value of the variable **i**. If **i** has value **10**, then the result of this command is **11**

[expr i + j]: **i** and **j** must be nodes in the local stack frame both must contain numbers. This returns their sum.

[expr $i + $j]: Produces the same answer as **[expr i + j]**

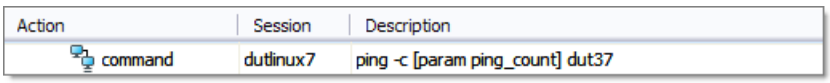
[expr (($packetsSent - $packetsRcvd)/$packetsSent)*100] is replaced by the packet loss percentage.

expr {[math.sin $i]/2}**: Notice that the syntax for the iTest **expr** command differs from the Tcl syntax: **expr {sin($i)/2}

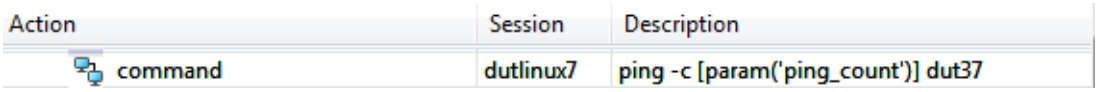
param command: Returning parameter values

Returns the value of a parameter (typically for insertion into a command or property).

In this example, the **param** command in the **command** step is inside a field replacement that is replaced just before the step executes. The command is replaced with the value of the parameter named **ping_count**. So, if the parameter had the value **9**, then the step would execute as **ping -c 9 dut37**. (Field replacements are described in Chapter 28, “Field Replacements”.)



Python:



About encrypted (masked) parameters

The **param** and **profile** commands never decrypt any parameter whose value is masked (typically, sensitive information like passwords). This ensures that the parameter value remains encrypted whenever the parameter is used in a test case or test report.

Accessing parameters defined in a session profile

Parameters can be defined in session profiles, response maps, testbed, for test case steps and overall test cases, and in a child test case executed by a **run** command. To access a parameter defined in the session profile associated with a step, use the **profile** command. See "profile command: Accessing parameters that are defined in session profile" on page 373.

Tip In any field that supports field replacements (including test case commands), the fastest way to insert a **param** command is to right-click where the command should appear and then select **Insert Parameter**. See "Inserting a parameter into a property or test case step" on page 741

Syntax

Tcl: param *parameterNameOrQuery* ?*defaultValue*?

Python: param(*'parameterNameOrQuery'*, *'defaultValue'*)

parameterNameOrQuery is the name of the parameter or of a query. If the parameter is a child of a node in a structure, then use *parentNode/paramName*

defaultValue: You have the option to specify a default value for the parameter if no value is found (for example, if the parameter inherits a blank **Value**).

Note The optional *defaultValue* argument is useful when you use the **param** command in a session profile property and you expect to start manual sessions. Because the parameter is normally obtained from the execution context, and manual sessions do not have an execution context, the command will use the specified default value when you start a manual session.

profile command: Accessing parameters that are defined in session profile

Returns the value of a parameter that is defined in the session profile associated with a particular session.

Note Parameters can be defined in session profiles, response maps, testbed, for test case steps and overall test cases, and in a child test case executed by a **run** command. To access a parameter defined in the session profile associated with a step, use the **param** command.

Tip In any field that supports field replacements (including test case commands), the fastest way to insert a **profile** command is to right-click where the command should appear and then select **Insert Parameter**. See "Inserting a parameter into a property or test case step" on page 741.

About masked (encrypted) parameters

The **param** and **profile** commands never decrypt any parameter whose value is masked (typically sensitive information like passwords). This ensures that the parameter value remains encrypted whenever the parameter is used in a test case or test report.

Syntax

Tcl: profile *session_ID parameter_name_or_query* ?*default_value*?

Example: *eval set a [profile s1 document/sessionProperties/@databaseType.inherit]*

Python: profile*('session', 'parameter_name_or_query', 'defaultValue')*

Example: *eval a = profile('s1', 'document/sessionProperties/@databaseType.inherit')*

session_ID is the session ID for the step (associated with the session profile that defines the parameter). You can use **.** (the period character) to mean "the session ID associated with the current step".

parameter_name_or_query is the name of the parameter or a query.

default_value: You have the option to specify a default value for the parameter if no value is found (for example, if the parameter inherits a blank **Value**).

Note The optional *default_value* argument is useful when you use the **profile** command in a session profile property and you expect to start manual sessions. Because the parameter is normally obtained from the execution context, and manual sessions do not have an execution context, the command will use the specified default value when you start a manual session.

query command: Inserting the results of a query

The **query** command returns the result of a query.

Tip In any field that supports field replacements, the fastest way to insert a query on a stored response is to use the context (right-click) menu. See "Applying queries to stored responses" on page 681.

Description

Returns the result of the query — a number, a string, or a list of strings. In the case of a list of strings, if an element is empty or contains whitespace, then it will be surrounded by **{ } (in Tcl) or [ ]** (in Python) characters automatically. Otherwise, the list of strings is just returned as a concatenation of the elements, separated by whitespace.

If "." is used for the location, then this is a special case and treated as pointing to the response to the current step. It is only valid for fields that are used after the step's execution is complete (typically in analysis rules).

If the node identified by **query** is not a response object (as identified by a corresponding attribute on that node), then iTest declares an error in the test report and on the Execution view.

Tip You can set preferences for queries. See "Setting preferences for response mapping" on page 601.

Syntax

Tcl: query ?**-alwayslist**? *varName mapperQuery*

Python: query(*'variable_name', 'mapper_query', alwayslist=False)*

Tcl	Python	Description
varName	*variable_name*	Variable that stores the response content. (Responses are stored using the **Store response in** property for a step). You can use "." to indicate the response for the current step.
mapperQuery	*mapper_query*	Mapping query that will be applied to the structured data in that response object. Either an XPath query or a query from a response map, as defined in the response map for the step. If **query** includes whitespace, it must be surrounded by double-quotes.
-alwaysList	*alwayslist*	The optional **-alwaysList** flag causes a single extracted value to be stored in a list with a single element, rather than in a scalar string. (A response with zero values or multiple values is always stored in a list.) This setting is important when you're using the response as the argument to a **foreach** statement and a single extracted value can contain whitespace. When you use the **-alwaysList** flag, a **foreach** statement that iterates over the stored variable will loop once for the match (rather than once for each word in the match).

Mapper queries support field substitutions, but some queries may also contain special interpreter characters. So you may need to "escape" these special characters. For example,

```
Tcl: [query myResponse inputPktsByPort("$portName")]
Python: query(myResponse inputPktsByPort('portName')
```

In this case, **$portName** or **portName** will first be substituted to become for example, **FastEthernet1/0/1**. In Tcl test cases, you must place double-quotes around **$portName** because the query engine is XPATH, which requires strings around its arguments.

For a more complicated query:

```
Tcl: table/row[1]/fieldB
Python: query("myResponse", "table/row[1]/fieldB")
```

In this case, the square brackets will not be appropriate for interpreter substitution, use:

```
[query myResponse table/row[1]/fieldB]
```

Examples Tcl

[query myResponse myQuery] Replaced with the result of applying the specified mapper **myQuery** against the stored response object. (A mapper query is a query that is auto-generated by a iTest response map.) To know which queries are valid, you can select the step whose response was stored or you can look in the Queries view or Structure view to select the query.

[query showInt inputPktsByPort('Fa1/0/1')] Goes to the response stored at **showInt** location, and performs query **inputPktsByPort(Fa1/0/1)**

[query . versionMinor] Uses the response for the current step and performs the query **versionMinor**.

Special case When certain events are fired during execution of a step (for example, **OnConnectionFailed**), actions may occur as a result. Some events fire during the execution process before mapping of the step has occurred. In those cases, **[query . myQuery]** will not work because the response has not yet been mapped.

response command: Accessing response data that is stored in a variable

To cause iTest to store the response to a step into a variable, you set the **Store response in variable** property for the step. By default, the variable is a complex object that includes both the text of the response and the structured data part of the response. (See "Storing a response into a variable (for use later in the test)" on page 140 for details.)

Now, to return the stored response content from the variable, use the **response** command. The **response** command can return the entire response text and structured data for analysis or it can return particular data from the response text (for example, return a list of matches by applying a regex to the response).

Note If you set the **Store only the text of the response** property for a step, then only the response text (and not the structured data part of the response) is stored in a simple variable.

In **Tcl** test cases, you cannot use a **[response** *varName***]** field replacement to return data. Instead, you must use **$***varName* or **[get** *varName***]** to return the full response text.

In **Python** test cases, for example, when using eval print("This string will be stored in a variable"), stores the response in a variable named "output".

eval response("output")

Do not check the option **Store only the text of the response**, in **Step Properties > Other Post-processing**.

Syntax

Tcl: response ?**-alwayslist**? ?**-group** *numberOrName*? *varName* ?**regex**?

? surrounding an argument means optional

Python: response*('varName', alwaysList=True, regex='', group='number | name')*

The optional **-alwaysList** (Tcl) or **alwayslist** (Python) flag is useful when you use the return data as the argument in a **foreach** statement. The flag causes a single returned value to be stored in a list with a single element, rather than in a scalar string. (A response with zero values or multiple values is always stored in a list.) This setting is important when you're using the response as the argument to a **foreach** statement and a single returned value can contain whitespace. When you use the **-alwaysList** (Tcl) or **alwayslist** (Python) flag, a

foreach statement that iterates over the stored variable will loop once for the match (rather than once for each word in the match).

The optional **-group** flag and *numberOrName* argument is a regular expression that defines something more specific to return from the response text.

varName/*variable_name* Is the name of the variable containing the stored response. If *varName* includes whitespace, it must be surrounded by double-quotes (which will be excluded from the location query before use).

The optional **regex** argument is a regular expression that defines something more specific to return from the response text.

Recommendations

In any field that supports field replacements, the fastest way to insert a **response** command is to right-click at the intended location and select **Stored Response**.

If the regular expression in the command does not use substitution, then surround it with **{ } (Tcl)** braces.

If there is a mismatched closing brace in the regex, then place double-quotes around the entire regex and place backslashes in front of all special characters (", [,], $, \) except where you actually want substitution.

Return values

A response with zero values or multiple values is always returned in a list.

You can specify whether to return a single value as a scalar string or as a list. See the description of the optional **–alwaysList** flag for recommendations on when to return a single value as a list.

Storing response data in a variable

See "Storing a response into a variable (for use later in the test)" on page 140.

Examples

Assume that we have stored a response in a variable called **myResponse**:

In an **eval** step, you can submit any of the following commands:

- **response myResponse** — Returns the entire response body
- **response myResponse {10.155.3.37}** where **10.155.2.37** is the regular expression

tcl command: Evaluating Tcl statements in the execution kernel's Tcl interpreter

The **tcl** command calls on the execution kernel's Tcl interpreter, evaluates the argument in the **Description** cell as a statement, and then returns the result of that evaluation. STDOUT and STDERR are not used.

Use the **tcl** command to have the iTest interpreter "communicate with" the execution kernel's Tcl interpreter.

```
[tcl {statement}]
```

tclexpr command: Evaluating Tcl expressions in the execution kernel's Tcl interpreter

The **tclexpr** command calls on the execution kernel's Tcl interpreter, evaluates the argument in the **Description** cell as an expression, and then returns the result of that evaluation. STDOUT and STDERR are not used.

Use the **tclexpr** command to have the iTest interpreter “communicate with” the execution kernel's Tcl interpreter. Contrast with the **expr** command.

```
[tclexpr {expression}]
```

call command: Call a procedure and get the return value from that procedure

Note Applies to **Tcl** test cases.

Use the command to call a procedure in the test case with optional arguments and values. The **call** command returns the result of executing the procedure, which is defined using 'return' or 'write'.

Note The procedure must return a value so that the substitution could be processed successfully, otherwise exceptions will occur.

Syntax

```
call testcase_path#procedure_name -arg value
```

testcase_path: is the uri of the test case that will be executed, typically starting with project://..

procedure_name: is the name of the procedure in the test case. Default will be main if not specified.

-arg value: is the series of arguments defined for the procedure.

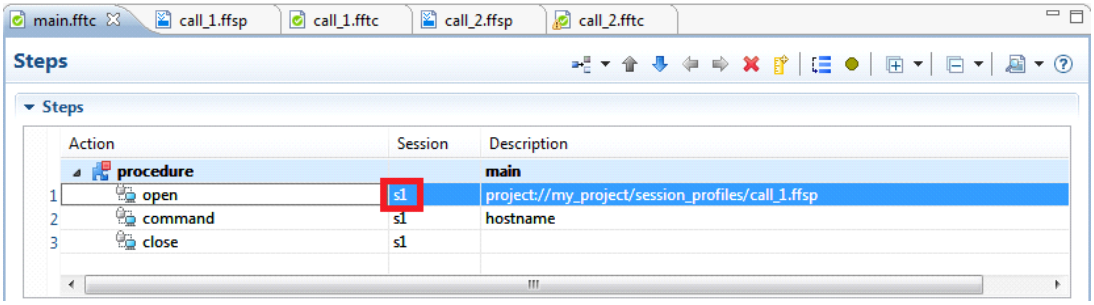
The procedure will be executed in a separate thread, and the main execution thread will wait for the procedure to be finished so that the return value can be retrieved.

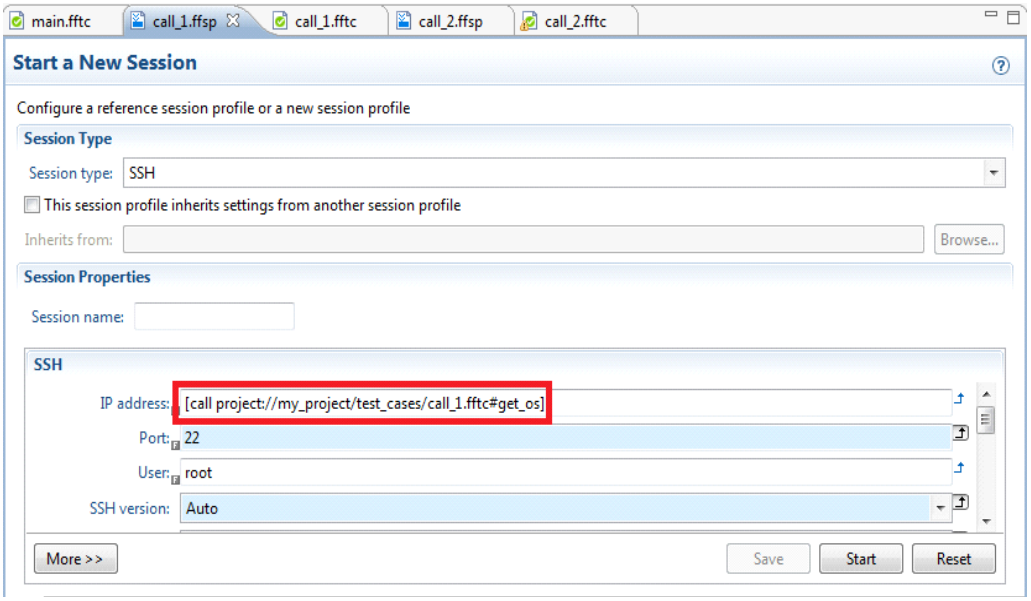
CAUTION To avoid the execution loop, make sure that the called procedure does not invoke the current procedure. If you use the call command in open step, for example, use the field substitution in session profile properties, make sure that the open step session id is different with the called procedure's open step session id, otherwise there will be an execution loop.

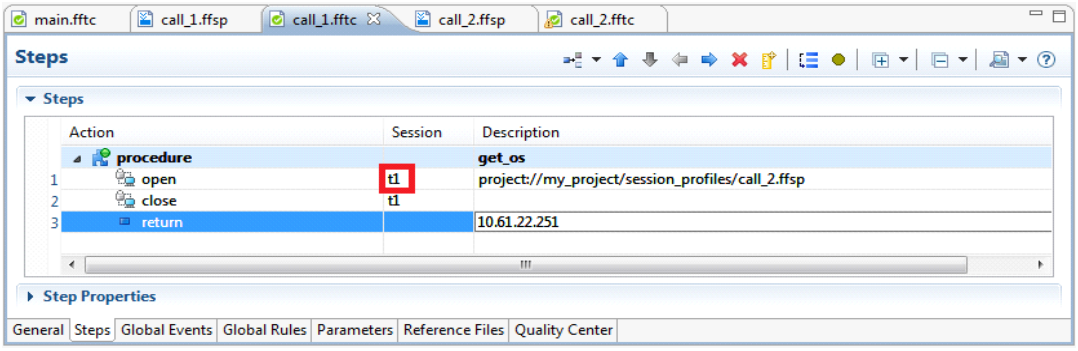
For example, in the following test cases (see illustration below), main.fftc opens a session profile **call_1.ffsp** whose IP address uses call command to get the value, and the **call_1.fftc** also has an open step, notice that these two open steps session ids are different: one is **s1**, the other is **t1**.

Examples

```
call project://my_project/test_cases/sample.fftc
call project://my_project/test_cases/sample.fftc#proc_a
call project://my_project/test_cases/sample.fftc#proc_a –arg1 value1 –arg2
value2
```







jsonSelect command: Get the node value from json string based on the query xpath

Use **jsonSelect** command to get the node value from the json_string, based on the provided query_xpath. The command returns the value of the specific node based on the provided xpath.

Note The xpath should be a valid xpath for XML (not the JsonPath - http://goessner.net/articles/JsonPath/), otherwise no value is returned. The provided json string will be converted to xml first, and then evaluated using the xpath and gets the specific value.

Syntax

Tcl:

```
jsonSelect json_string query_xpath
```

json_string is the valid json string.

Query_xpath is the valid xpath for xml.

Python:

```
jsonSelect('jsonString', 'queryXpath')
```

json_str is the valid json string.

xpath is the valid xpath for xml.

Example: eval jsonSelect("{'key1':'value1'}", "key1")

Note If the xpath location is not a single value, then an assembled version of sub values will be retrieved.

Examples

Tcl:
{"array":[1,2.4,"null1":null,"i":"j","k":"l"}],"boolean":true,"new":"notNull","number":123,"object":{"a":"b","c":"d","e":"f"},"string":"Hello World","null2":null}

Python:

eval
jsonSelect("{'array':[1,2.4,{'null1':'None','i':'j','k':'l'}],'boolean':'True','new':'notNull','number':123,'object':{'a':'b','c':'d','e':'f'},'string':'Hello World','null2':'None'}", "array/item[3]/i")

xpath	value	comment
number	123	-
array/item[1]	1	The array type in json will be converted to xml string with an additional "item" key.
array/item[3]/i	j	
object/c	d	-
object	bdf	The xpath "object" represents an object type in json, thus we can only get the values based on the converted xml contents.

Commands that return information about testbeds

Note Not applicable for Python test cases.

Commands that return information about testbeds

testbed commands return information about the testbed devices used during execution.

Tip Be sure to see the **info testbed** command, as described in "'info' subcommands for directories, URIs, and workspaces" on page 359.

'testbed devices' command: List device names

Returns a list of testbed device names.

Syntax

Tcl: testbed devices

Return values

Returns the list of testbed device names from the testbed used by the test case.

Returns an empty list if there is no testbed or the testbed is empty.

'testbed *deviceName propertyName*' command: Get the value of a device property

Returns the string value of the specified property for the specified testbed device.

Syntax

Tcl: testbed *deviceName propertyName* ?*defaultValue*?

Python: testbed('*deviceName*', '*propertyName*', '*defaultValue*')

Arguments

deviceName must be a valid testbed device name

propertyName depends on the session type

defaultValue (optional) During execution, if a value for *propertyName* does not exist (because the device does not support the property), then the command returns *defaultValue*.

Return values

For single-value properties, returns the string value of the specified property for the specified device.

For multi-value properties, returns a string list

- If the property is encrypted, then the encrypted value is returned.
- If the property value specified by *propertyName* is empty (blank), then an empty string is returned.
- If the property has field replacements enabled, then substitution is performed at runtime.
- If a value for *propertyName* does not exist, then the command returns *defaultValue*. If *defaultValue* is not specified, then an error is generated.

Adding a testbed command

See "Inserting a 'testbed' command as a field replacement" on page 382.

Error conditions

- If a value for *propertyName* does not exist, and *defaultValue* is not specified, then an error is generated.
- Error generated if testbed is not found
- Error generated if device is not found

Inserting a 'testbed' command as a field replacement

A **testbed** command (of the form [**testbed** *deviceName propertyName* ?*defaultValue*?] in Tcl or **testbed***(deviceName, propertyName defaultValue)* in Python) returns the value of a specified property for a specified testbed device (in the testbed used by the test case).

Use the **Insert Testbed Device Property** dialog box to insert a **testbed** command in the proper format into a property setting or into a step. The command is placed within **[** and **]** brackets to make it a field replacement.

1. Place the cursor in the desired position in the **Description** cell or the property field; alternatively, select text to replace. Now right-click and select **Insert > Testbed Device Property**.
2. On the **Insert Testbed Device Property** dialog box, select a device from the **Testbed Devices** list. The **Device Properties** list is updated with the properties for the device you selected.
3. Select a property from the **Device Properties** list. The value of the selected property appears in the **Property value** text box. The text in the **Field replacement** text box is updated with a **testbed** command that reflects your choices.

Note You have the option to edit the text in the **Field replacement** text box. In particular, you can specify a default value for the property — the value to use if the **testbed** command cannot determine the value of the property. Add the default value after the property name.

---

4 Click **Insert**. The wizard inserts a **testbed** command in the following format (this example includes a default value):

```
Tcl: [testbed deviceName propertyName defaultValue]
```

```
Python: testbed('deviceName', 'propertyName', 'defaultValue')
```

For example,

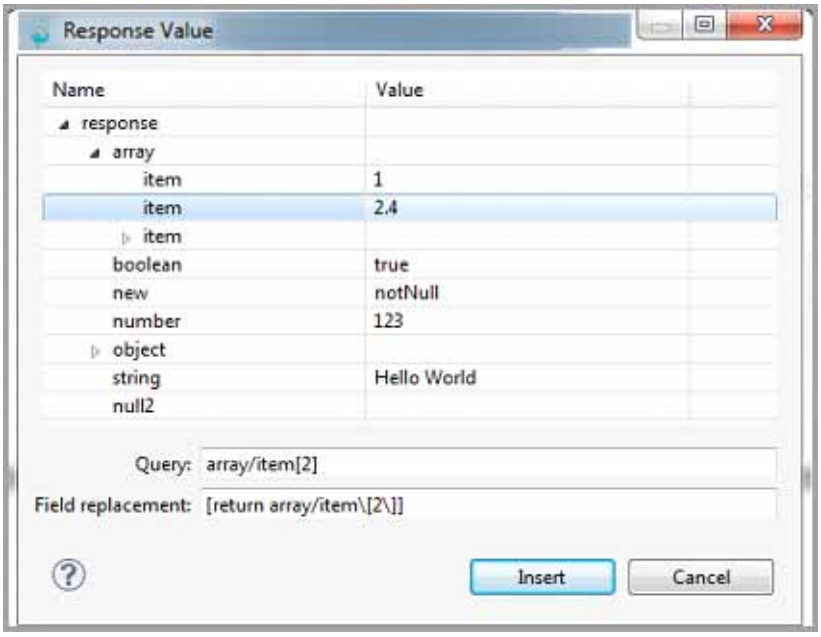
```
[testbed ssh_device userID Amish]
```

```
testbed('ssh_device', 'userID', 'Amish')
```

Commands that return sample json xpath used in Store Processor

A return command ([**return query_xpath**]) returns the xpath for a sample JSON response. The return value will be used in the store processor, which replaces the evaluated values with the extracted values.

Use the **Insert->Response Value menu** to open the **Insert Return Value** dialog to get xpath value.



CHAPTER 18

Scheduling Execution

Scheduling a job

You can schedule tests to execute as a group in the sequence that you specify. You first define one or more sets of test cases to execute. You then define the list of test cases/test suites to schedule for execution: the *job*.

Defining a job (using the Schedule wizard)

Step 1 Specify the tests to run and the Topology and/or Parameter file

1. On the iTest Explorer, right-click and select Job. The **New Job** wizard opens to the **Job** (create a new job file) page.
2. Select folder to create the new job and click **Next**.
3. Specify the test case/rest suite, on the **General** (Specify items to execute) page.
   a. Navigate to the appropriate project or folder and select as many test cases/test suites as needed (Ctrl+click, Shift+click) and then click **OK**.
      - Test cases/suites are executed in the listed order. To move a test case/test suite, select it and use the up and down arrows.
      - To remove a test case/suite from the list, select it and then click  .
4. You have the option to specify the **Topology** and/or **Parameter** file to use for all test executions. The files that you specify here override the settings in the test cases.
5. Click **Next**.

Step 2 Specify the schedule for execution

On the **Schedule Job** page:

Once

Select **Once** to run the job a single time at the specified time. Specify the following settings:

Immediate	Run the job as soon as you click **Finish**. This setting ignores the time specified by the **Start at** property.
Start at	Specify the hour, minute, and date for the run. Click any time value and change it using the up and down arrows.

Recurring

Select **Recurring** to run the job on an hourly basis or once per day every day or on specified days of the week over a specified time period.

To execute a test case multiple times per day, create a job for each execution time.

Specify the following settings:

Start at	Specify the time of day that the job should start. Click **Noon** or **Midnight** to set that time. **Tip** Click **Noon** or **Midnight** and then use the arrows to fine-tune the time. **Note** U.S. users: The start time will fail for the one hour during which the time changes due to daylight saving time. For example, between 2 AM and 3 AM on 14 March because the time does not exist.
Repeat **Hours:minutes**	Check the box to execute on an hourly basis. Uncheck to execute on a daily or less frequent basis. Specify the time between executions.
Stop after	If you checked **Repeat every**, then specify the time when execution should stop. **Tip** Click **Noon** or **Midnight** and then use the arrows to fine-tune the time.
Days	Specify the days of the week that the runs should occur. Use the **All** and **None** links to check and clear all days.
End date	Specify the date in the future that scheduled runs of the job should stop.

6 If you specified **Immediate** run, then click **Finish**. The run will start in a moment.

Note By default, the last page of the wizard enables the job to run as scheduled, so there is no need to continue past this wizard page for an immediate run.

7 For jobs with start times other than **Immediate:** On the **Enable Job** page, you specify whether the job that you just defined should become an actively scheduled job or should wait to start its scheduled runs until you enable it at some later time.

Check **Enable this job** to allow the job to run on the schedule that you just configured. This is the default setting.

Step 3 Optional: Specify additional settings

Each setting on the **Additional Settings** page is optional. Specify settings as needed and then click **Next**.

Tags Optional: Specify database tags that should be associated with test reports.

- Tags are used only if test reports are saved to an external database.
- Each report is tagged with the text.
- You can use the value to search for test reports in the database.

Specify tag settings using **name=value** pairs that are separated by commas. For example, **Environment=Linux, Type=Stress**

Note Tags are not used with the iTestRT **--comparereports** option.

Email To / Email cc Optional: Specify the email addresses (both **To** and **cc**) that should receive notification when test case execution finishes. If you type more than one address in a text box, then separate the addresses using commas.

To modify the schedule for a job

1. Delete the existing job
2. Create a new job that specifies the same Test Case/test Suite as the original with the new schedule settings.

To cancel a job (cancel execution)

In the Execution Activity view, right-click the job and select **Cancel**. The job is removed from the queue.

Execution Activity view

When you click **Finish** in the **Schedule** wizard, the scheduled job appears in the Execution Activity view. Use the view to monitor test case/test suite executions that you have scheduled and that have executed.

To access a test reports for a test case that has executed, right-click the entry and select **View Report**.

[Test Case]	The first column lists the test case/test suite
Status	PENDING – Job received but not yet dispatched to execution agent SCHEDULED – Job dispatched to an execution agent COMPLETE – Execution completed without errors ERROR – A problem occurred during execution
Job Name	The name that was specified for the job when it was defined.
Originator	The person who submitted the request
End Time	The timestamp when the job was completed.

Overview: Scheduling execution using Jobs

You can schedule tests to execute at a specified time in the future or to execute on a recurring basis once per day at a specified time on specified days of the week. Your definition of the times and days for execution and the test or test suite to execute is called a *job*. A job can define any of the following kinds of execution schedule:

- Define a job to execute the test case or test suite while you attend a meeting.
- Schedule a nightly test suite run to make sure that regression catches any bugs.
- Schedule a weekly early morning test suite run. You specify a maximum time limit to ensure that the testbed frees up before other developers need it later in the morning.

- You can *disable* a job at any time. This means that the job definition still exists, but is taken out of the schedule (the job will not run) until you place it back into the schedule (by *enabling* it).

Important note for Linux users

On some Linux systems, Java applications may not pick up the correct time zone. You might experience the problem as scheduled jobs executing at incorrect times. To eliminate the problem, set the following environment variable before you start Spirent iTest:

```
export TZ=`cat /etc/timezone`
```

iTestRT

The iTestRT executable is **itestrt**

While iTestRT runs a job, it sends output from the job to STDOUT.

iTestRT output is updated each time a job starts or finishes execution. iTestRT displays the current snapshot of all scheduled jobs: running jobs and jobs that are waiting for execution.

iTestRT continues running until the specified job has no more work pending. For recurring jobs, this means that itestrt will run indefinitely.

For full details, see “Running a job using iTestRT” on page 392.

iTestCLI

iTestCLI does not support jobs.

Spirent iTest

Spirent iTest runs jobs in the background, so CLI sessions do not appear in execution views. In contrast, browser-based sessions (for example, Web and Swing) display the browser activity in the session window during execution.

When Spirent iTest starts up

- When Spirent iTest opens, it starts any enabled jobs at the scheduled time.
- Jobs whose start times had elapsed while Spirent iTest was not running (and therefore were not run) are started at the next scheduled time. The missed run is ignored.
- Disabled jobs are ignored.

Exiting Spirent iTest while jobs are running or scheduled to run

If you try to exit Spirent iTest while a job is running, a dialog box prompts whether to cancel execution and quit Spirent iTest or to continue running Spirent iTest so that execution can continue.

If you exit Spirent iTest while jobs are scheduled to run, then Spirent iTest exits. When you restart Spirent iTest, the process of running scheduled jobs resumes. Spirent iTest does not run jobs whose start time was missed while Spirent iTest was not running.

Monitoring job runs

See “Monitoring job runs” on page 393.

Configuring a job: The Job wizard

Use the Job wizard to add and modify a job definition.

1 Start the **Job** wizard using one of the following methods:

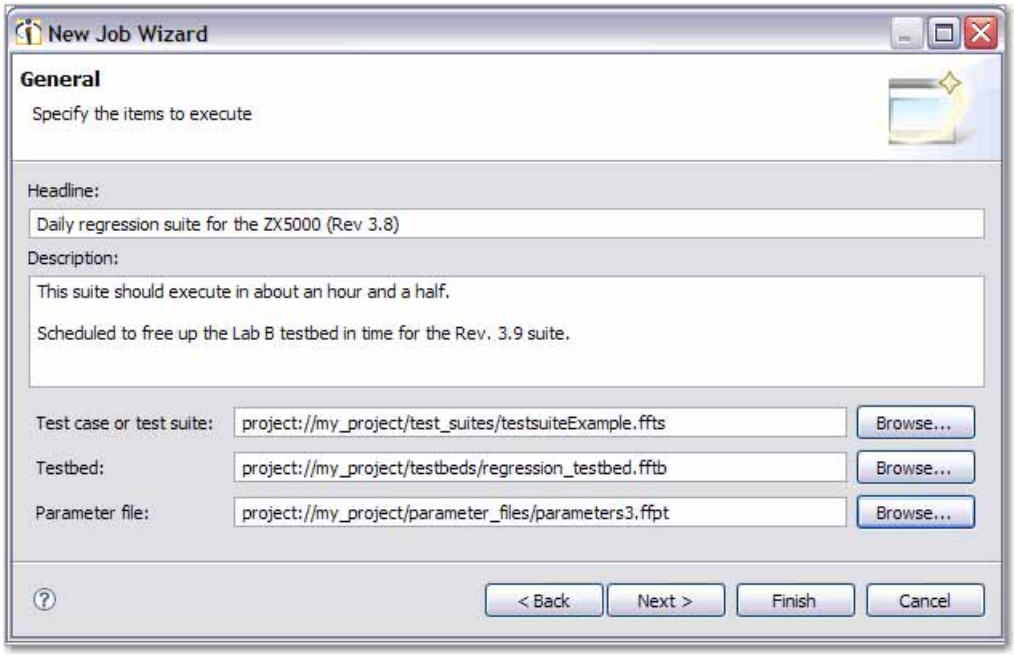
- If you know the test case or test suite that the job should run, then right-click it in the iTest Explorer or Favorites view and select **New Job**.
- While working in the **Job** editor, click **New Job** .
- Click **New** and select **Other**. (Alternatively, click **File > New > Other**.) The **Select a wizard** page opens. In the **iTest** group, select **Job** and then click **Next**.

2 The **Job** wizard starts. On the **Job** page, specify the following properties and then click **Next**.

Enter or select the parent folder	Select the project and folder that will hod the new job file.
File name	Provide a name for the job that you are creating. This is the name that will appear in the Jobs view.

Advanced users To define a job in the file system outside the current workspace, click **Advanced** and browse to the location. To specify a path variable that will help Spirent iTest to locate the resource, click **Variables** and then specify the **Name** and **Location** of the path variable.

3 On the **General** page, you configure identifying information for the job and specify the documents to use during execution.



Headline	Optional. Type a single line of text that describes the job that you are scheduling. This text appears on the Job editor to help you and coworkers when working on the job.
Description	Optional. Type text that will help coworkers understand the intent and operation of the job.
Test case or test suite	Specify a test case or a test suite to execute. All test cases in a test suite will execute.
Topology or Testbed	Optional. Specify a topology or testbed file to use during execution. The topology or testbed that you specify will be used during execution instead of any topologies or testbeds that are referenced by the test case or test suite.
Parameter file	Optional. Specify a parameter file to use during execution Using a parameter file enables you to supply values that a test case should use at runtime. See Chapter 33, “Parameters”.

4 On the **Schedule Job** page,

Once

Select **Once** to run the job a single time at the specified time. Specify the following settings:

Immediate	Run the job as soon as you click **Finish**. This setting ignores the time specified by the **Start at** property.
Start at	Specify the hour, minute, and date for the run. Click any time value and change it using the up and down arrows.
Enforce time limit	To ensure that the job does not exceed a particular run time, check the box and specify the maximum time allowed for the job to finish its run. If a test is executing when the limit is reached, then iTest aborts execution and stops running the job. The Test case result is set to **Aborted**. The default setting of unchecked box and a time of 00:00 mans that there is no time limit for the run.

Recurring

Select **Recurring** to run the job on an hourly basis or once per day every day or on specified days of the week over a specified time period.

To execute a test case or test suite multiple times per day, create a job for each execution time.

Specify the following settings:

Start at	Specify the time of day that the job should start. Click **Noon** or **Midnight** to set that time. **Tip** Click **Noon** or **Midnight** and then use the arrows to fine-tune the time. **Note** U.S. users: The start time will fail for the one hour during which the time changes due to daylight saving time. For example, between 2 AM and 3 AM on 14 March because the time does not exist.
Repeat every Hours:minutes	Check the box to execute on an hourly basis. Uncheck to execute on a daily or less frequent basis. Specify the time between executions.
Stop repeating after	If you checked **Repeat every**, then specify the time when execution should stop.
Days	Specify the days of the week that the runs should occur. Use the **All** and **None** links to check and clear all days.
End date	Specify the date in the future that scheduled runs of the job should stop.
Scheduled jobs	The table displays the first hundred scheduled runs as you have specified them using the property settings described here. The table auto-updates whenever you change a setting.
Enforce time limit	To ensure that the job does not exceed a particular run time, check the box and specify the maximum time allowed for the job to finish its run. If a test is executing when the limit is reached, then iTest aborts execution and stops running the job. The Test case result is set to **Aborted**. The default setting of 00:00 mans that there is no time limit for the run.

5 If you specified **Immediate** run, then click **Finish.** The run will start in a moment.

Note By default, the last page of the wizard enables the job to run as scheduled, so there is no need to continue past this wizard page for an immediate run.

6 For jobs with start times other than **Immediate:** On the **Enable Job** page, you specify whether the job that you just defined should become an actively scheduled job or should wait to start its scheduled runs until you enable it at some later time.

Check **Enable this job** to allow the job to run on the schedule that you just configured. This is the default setting.

Disabling or enabling a job

When you create a job, it is ***enabled*** by default—the job will run on the schedule that you configured. When you ***disable*** a job, you remove it from the execution schedule, but do not delete the job definition file. To cause the job to execute on schedule once again, you enable it.

Enabling a job

In the iTest Jobs view, right-click the job and select Enable.

Disabling a job

If the **Job Status** is **Waiting**, right-click it and select **Disable**.

If job is currently executing (its **Status** is **Running**), you must first cancel the execution and then disable the job.

1. Right-click the job and select **Cancel**.
2. Right-click the job and select **Disable**.

Deleting a job file

- **To delete a job (remove the ffjd job definition file):**

Right-click the job in the iTest Explorer and select **Delete**. The file is deleted and the job no longer appears in either the iTest Explorer or the iTest Jobs view.

Running a job using iTestRT

To run a job from the command line, you enter a command of the following form:

```
itestrt --job [jobUri]
```

Note The full iTestRT command set appears in “iTestRT command reference” on page 709.

- iTestRT starts and writes the following information to the console:
  - If the job is running, iTestRT displays **Running job: <jobFilename>** and the sort of information shown in “Example iTestRT output” on page 393.
  - If iTestRT is waiting for a job to start, it displays **Waiting for job** and a list of scheduled jobs (five maximum) with the next start time and time limit. STDOUT is then silent until the next job starts running.
- While iTestRT runs a job, it sends output from the job to STDOUT.
- When the run completes, then iTestRT displays the scheduling (if any) of the next run.
- iTestRT continues running until the specified job has no more work pending. For recurring jobs, this means that iTestRT will run indefinitely.
- To kill iTestRT at any time, press Ctrl-C.

Tip Use the iTestRT reporting plug-ins to publish reports in various ways.

Example iTestRT output

iTestRT output is updated each time a job starts or finishes execution. iTestRT displays the current snapshot of all scheduled jobs: running jobs and jobs that are waiting for execution.

Tip Use the **--job.quiet** option to suppress output from job execution.

```
1. Loading file:/c:/itestRT/workspace/my_project/jobs/new_job.ffjd - OK
        new_job (Recurring on MTWTFSS at 1:49 PM until 12:00 AM Mar 18, 2009)

Starting: new_job (Recurring on MTWTFSS at 1:49 PM until 12:00 AM Mar 18, 2009)
Running jobs:
#   Info                                                                    Start Time             Elapsed Time
-------------------------------------------------------------------------------------------------------------
1   new_job (Recurring on MTWTFSS at 1:49 PM until 12:00 AM Mar 18, 2009)   1:49 PM Mar 13, 2009   0s


Scheduled jobs:
<None>


Finished: new_job (Recurring on MTWTFSS at 1:49 PM until 12:00 AM Mar 18, 2009)
    Status: ERROR
Running jobs:
<None>

Scheduled jobs:
#   Info                                                                    Next Start Time
-------------------------------------------------------------------------------------------
1   new_job (Recurring on MTWTFSS at 1:49 PM until 12:00 AM Mar 18, 2009)   1:49 PM Mar 14, 2009
```

Monitoring job runs

Scheduled Jobs view

The Scheduled Jobs view displays the list of jobs that are scheduled to be run.

Job Name	The name that was specified for the job when it was defined.
Next Execution Time	The time when the next execution of the job will occur
Originator	The person who submitted the request

iTest Jobs view

The iTest Jobs view displays the list of existing jobs (enabled, disabled, and currently running). For each job that has run, the view lists the run and all test reports associated with the test cases that executed during the run.

For full details, see “iTest Jobs view: Monitoring and managing job schedules and results” on page 394.

Progress view

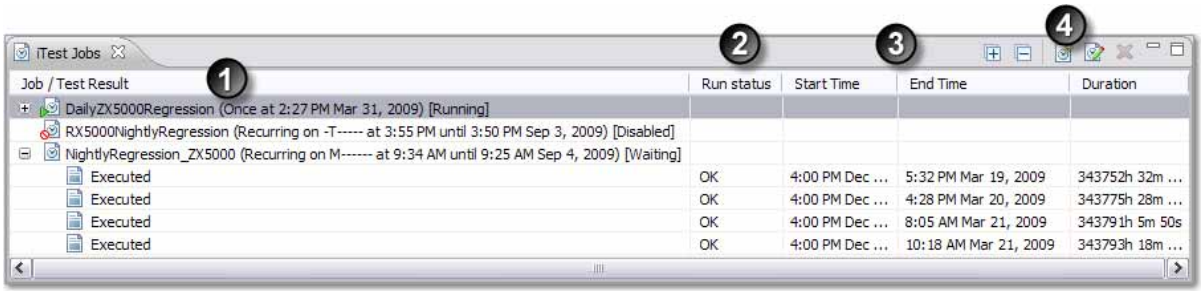
The Progress view displays the list of currently running jobs. Use either of the following methods to open the Progress view:

- Right-click in the Jobs View and select **Show Progress View**.
- Click **Show View > Other**. In the **General** group, select **Progress**.

iTest Jobs view: Monitoring and managing job schedules and results

The iTest Jobs view displays the list of defined jobs and their **enabled/disabled/running** status. For each job that has run, the view lists the run, its status, and all test reports associated with the test cases that executed during the run.

Click **Show View > iTest > iTest Jobs** to open the view.



1 Job / Test Result column

The **Job / Test Result** column lists all defined jobs.

- For each job, a summary of the schedule appears in parentheses and text and an icon indicates **Running/Enabled/Disabled** status. Under each job, the view lists the runs that have occurred.
- Double-click a job to open it in the **Job** editor.
- Right-click a job to:
  - Cancel the job (if the job is running).
  - Enable or disable the job (if the job is not running)
  - Modify the job configuration. If a job is running, you must first cancel it to modify its configuration.
  - Create a new job
  - Open the Progress view to display the progress of executing jobs
- For each completed run, the view lists the **Pass/Fail/Indeterminate/Abort** execution result and an associated icon.

Here is the structure of the information for a job that executes three tests (to expand or collapse entries, use ⊞ and ⊟ or use **Expand All** ⊞ and **Collapse All** ⊟ in the toolbar).

JobName (Summary of schedule) [Running / Enabled / Disabled]

Run 1 (on Monday, for example)

Test report 1A

Test report 1B

Test report 1C

Run 2 (on Tuesday, for example)

Test report 2A

Test report 2B

Test report 2C

2 Run Status column

The **Run Status** column indicates the status for the run of the job.

3 Start Time, End Time, and Duration columns

The **Start Time**, **End Time**, and **Duration** columns display the information for each test execution.

> **Tip** Click any column heading to sort.

4 The toolbar

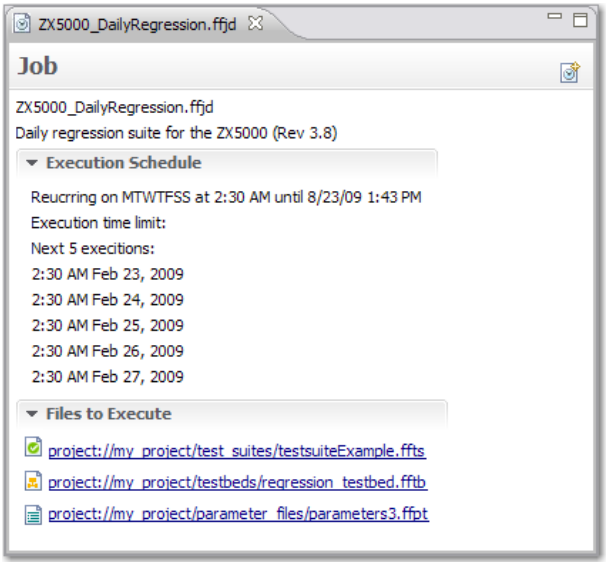
	Expand all or collapse all jobs in the view to display or hide the runs and test reports.
	**New Job**: Create and configure a new job
	**Modify Job**: Start the **Job** wizard so you can edit the configuration settings for the selected job.
	Delete the selected test report lines from the view. (Use Ctrl-click and Shift-click to select multiple items.)

Job editor

The Job editor provides an overview of a job definition. The view displays:

- The name and headline (brief description) of the job
- The upcoming schedule for the job
- The files that will be used during each run. To open any of the files in the appropriate editor, click the link

To modify any of the job settings, click **Modify Job**. The **Job** wizard opens to the **Job** page, as described in Step 2 of “Configuring a job: The Job wizard” on page 389.



Opening the Job editor

In the iTest Explorer, right-click a job and select **Open**. Alternatively, double-click the job.

Viewing and managing jobs using Spirent iTest Explorer

Creating a new job

To create a job that should execute a particular test case or test suite, then right-click the test in the iTest Explorer or Favorites view and select **New Job**.

Managing existing jobs

In the iTest Explorer, right-click a job and select one of the following options:

- **Open in iTest Jobs View** — Open the iTest Jobs view and select the job. See “iTest Jobs view: Monitoring and managing job schedules and results” on page 394.
- **Open** — Open the job in the **Job** editor, as described in “Job editor” on page 395. Alternatively, double-click the job.
- **Modify Job** — Start the Job wizard on the **Job** page, as described in Step 2 of “Configuring a job: The Job wizard” on page 389.

Setting preferences for Spirent iTest jobs

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Jobs**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”

Open / close session windows during job runs	This setting does not affect test execution. It affects only what you see while test cases are executing. Check the box to make the opening and closing of sessions windows visible while test cases are executing. Uncheck the box to run the job without displaying session windows. Default: unchecked
Export test reports for test cases executed by jobs	Check the box to cause iTest to export a test report document in HTML format after each test case execution • One report is generated for each test case execution • Test report files are named with the test case name and a timestamp • Optional. You can specify the location to save reports using the **Folder for test reports** property • Optional. You can specify the location of XSL format files using the **Location of test report format files** property If you uncheck the box, then iTest does not automatically export test report documents. Default: unchecked

Folder for test reports	Optional. Used only if you check the box for **Export test reports**. Specify where to save the test report files. You can use any of the listed { } replacements in the URI that you specify. For example: `Project://my_project/jobs/{jobfilename}/reports_{date}` This text dynamically places the name of the **job.ffjd** file under **my_project/jobs/** and the **report** subfolder will include the date that the particular job ran.
Location of test report format files	Optional. Used only if you check the box for **Export test reports**. Specify the URI of the folder where the XSL format files (to apply to the test reports) are stored.

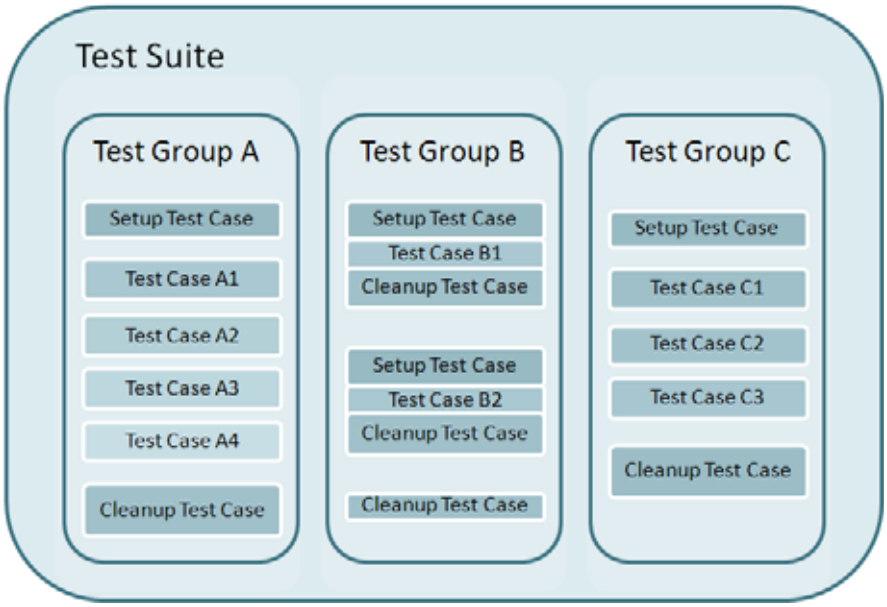
CHAPTER 19

Test Suites

Overview: Test suites

Once you define all of the test cases that thoroughly test a device or application, you might want to schedule them for regression testing — all tests to execute one after the other or in parallel. iTest provides the Test Suite editor to simplify the task.

A test suite contains one or more groups of tests. Each test group contains one or more test cases. When you define a test suite, you can specify that each test group should execute in a specified order or concurrently (in parallel). You use the Test Suite editor to specify which test cases should be members of each test group and which test groups make up a test suite. The test suite also specifies the order of execution, timeout limits, and whether setup or cleanup test cases should execute before and after each test case execution and/or before and after each test group.



Creating a test suite: The Test Suite wizard

The Test Suite wizard helps you to quickly create a new test suite.

1 Create the test suite document: Click **New** and select **Test Suite** and then click **Next**.

2 On the **New Test Suite** page of the **Test Suite** wizard, specify the following properties and then click **OK**.

Container	Path to the new test suite file. Default: **/my_project/test_suites**
File name	Provide a name for the test suite that you are creating. This is the name that will appear in the iTest Explorer and Favorites view.

3 On the **Root Folder** page, specify the following properties and then click **OK**.

Group root folder	Browse to the folder that contains the tests that you want to include in the test suite.
Filter files based upon matching criteria	To add test cases to the suite on the basis of filename, check the box. On the next wizard page, you will specify a filename filter so that only the test cases in the **Group root folder** whose filenames match the filter you specify are added to the test suite, then click **Next**. For example, you might specify a filter **Pattern** of ***regression*** to add only tests with the text "regression" in the filename. To add test cases to the suite on a basis other than filename, do not check the box and then click **Finish**. **Note** If you choose not to check the box, but later want to apply a filter in defining a test group, you later can apply a filter in the Test Suite editor. The editor enables you to define other types of filters: • **File match** (default): The **Pattern** is compared with the names of the files in the specified subfolder. Match with the filename. You can use the * wildcard character. • **Description match**: Match text in the **Description** property string for the test case. • **Owner match**: Match text in the **Owner** property string for the test case (typically, a person's name). • **Parameter match**: Specify a parameter setting Include or exclude test cases with a matching parameter setting. (for example, **testType=stress**). • **Parameter assertion**: Specify a **<paramName = value>** assertion. Include or exclude test cases for which the assertion is true.

4 If you checked the **Filter files based upon matching criteria** option, then the wizard opens the **Configure Filter** page. Specify the settings as described in the table and then click **Finish.**

Note All test cases starting with "_" (the **_setup** and **_cleanup** test cases) are excluded from the test case filtering process.

Pattern	Type the text that should be used as the basis for filtering test cases. For example, you might specify a **Pattern** of ***regression*** to add only tests with the text "regression" in the filename.
Match type	**Strict**: The filename must exactly match the text specified for the **Pattern**. **Regular Expression**: Interpret the text specified for the **Pattern** as a regular expression when comparing the filename to the **Pattern**. **Wildcard**: (default) The filename must match the text specified for the **Pattern**.

On match	Take the specified action when a test case's filename matches the pattern: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On no match	Take the specified action when a test case's filename does not match the pattern: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
Root folder	Browse to the folder that contains the tests that the filter should be applied to.
Allow subfolders	Check the box to search subfolders when applying filters, Default: Checked

The wizard saves the new test suite document (**.ffts** filename extension) and then opens it in the Test Suite editor.

Configuring setup and cleanup test cases for folders

Often, you need to run setup procedures before executing all tests in a folder and/or before each test. Similarly, you might need to run cleanup procedures after tests. Test suites support all such cases.

Note To specify that setup/ cleanup should occur for a test group, see "Global Setup / Cleanup properties" on page 412.

To enable this feature:

- Check the **Run setup/cleanup tests per folder** and/or **Run setup/cleanup tests per test** properties, as described in "Sorting properties: Specify the order of execution" on page 412.
- Name the setup/cleanup test cases as described here .
  - **Setup before executing all the test cases in a directory:** If a directory includes a test case named **_setup**, then it is executed before executing any of the test cases in the directory (or its subdirectories). If there is also a **_setup** test case in a subdirectory, then it also runs before the test cases in its directory (and further subdirectories).

    Note The top-most **_setup** always executes first, even if the first test case eligible for execution is in a subdirectory.

  - **Setup before executing each test case in a directory:** If a directory includes a test case named **_setup_each**, then it is executed before *each* eligible test case in its directory (including subdirectories). If there is a **_setup_each** in a child directory, then it executes after the **_setup_each** test case in its parent directory but before each test in its directory.

- **Cleanup after executing all the test cases in a directory:** A test case named **_cleanup** behaves in a similar way as **_setup**, but after all test cases have been executed and after its child directories.
- **Cleanup after executing each test case in a directory:** A test case named **_cleanup_each** behaves in a similar way as **_setup_each**, but after each eligible test case in its directory (including child directories). For test cases in child directories, it executes after any **_cleanup_each** in the child directory.

Note All test cases starting with "_" are excluded from the test case filtering process.

Example execution sequence using all options

_setup
folder1/_setup
folder1/folder2/_setup
folder1/_setup_each
folder1/folder2/_setup_each
folder1/folder2/testcase1
folder1/folder2/_cleanup_each
folder1/_cleanup_each
folder1/_setup_each
folder1/folder2/_setup_each
folder1/folder2/testcase2
folder1/folder2/_cleanup_each
folder1/_cleanup_each
folder1/folder2/_cleanup
folder1/_cleanup
_cleanup

Tip You might want to perform comparatively “bigger” setup/cleanup operations at the higher levels and smaller, faster setup/cleanup at the lower levels and/or in individual setup/cleanup procedures.

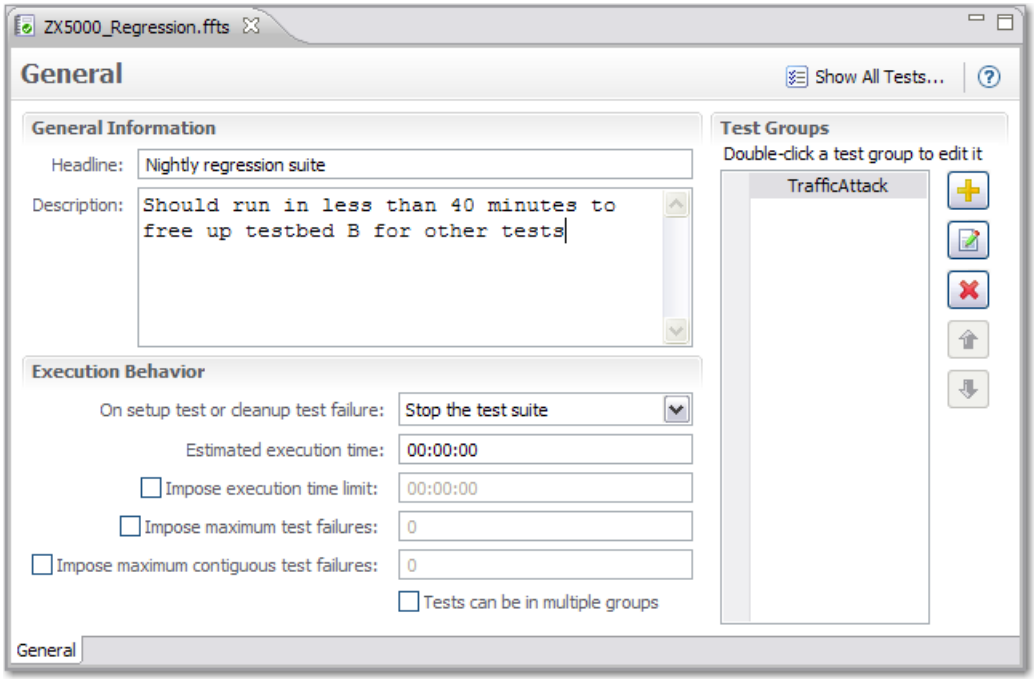
Configuring a test suite: The General page of the Test Suite editor

Use the Test Suite editor to configure and edit test suites. The primary tasks that you accomplish in the editor are to define test groups and to specify the criteria that determine which test cases should be included in each group. You also specify which setup and cleanup test cases should run and whether they should run for the group or for each test case.

Tip Whenever the Test Suite editor is selected, the **Test Suite** menu appears in the menu bar. The options that appear in the menu vary depending on the editor page in use.

1 Create a test suite as described in “Creating a test suite: The Test Suite wizard” on page 399. The wizard saves the new test suite document and opens it in the **General** page of the Test Suite editor. To edit an existing test suite, double-click it in the iTest Explorer.

2 On the **General** page, you specify the properties that define the test suite and create new test groups.



Tip Undo/redo (Ctrl-Z, Ctrl-Y) applies to any change.

General Information

Headline	Optional. Type a one-line description of the filter. For example, **Includes only regression tests for RevD sorted by filename** This text appears in the Favorites view for the test suite.
Description	Optional. Type text that describes the test suite to make its usage clear to coworkers.

Execution Behavior

On setup test or cleanup test failure	Specify the action to take when any setup or cleanup test fails for any reason while the test suite is running. **Stop the test suite**: iTest aborts setup or cleanup test execution and stops running the suite. **Stop the test group**: iTest aborts setup or cleanup test execution and stops running the group. The next group in the suite then begins running. **Continue**: iTest aborts setup or cleanup test execution and executes the next test (either in the current group or in the next group — whichever test is next). **Note** You can make use of special setup and cleanup test cases for test suites. See “Configuring setup and cleanup test cases for folders” on page 401.
Estimated execution time	This estimate is for how long it will take for the suite to run is obtained by summing the estimated execution time for each test cases in the test suite. The value for each test case is taken from the **Estimated execution time** property as specified on the Test Case editor **General** page for the test case.
Impose execution time limit	Optional. To ensure that the test suite does not exceed a particular run time, check the box and specify the maximum time allowed for the suite to finish its run. If a test is executing when the limit is reached, then iTest aborts test execution and stops running the suite. The Test case result is set to **Aborted**. The default setting of 00:00 mans that there is no time limit.
Impose maximum test failures	Optional. Specify the number of test case failures that you will allow to occur. If the limit is exceeded, then iTest aborts test execution and stops running the suite.
Impose maximum contiguous test failures	Optional. Specify the number of test case failures that you will allow to occur in a row (that is, one after the other with no successful executions between). If the limit is exceeded, then iTest aborts test execution and stops running the suite.
Tests can be in multiple groups	Check the box to allow any test that passes a filter to become a member of any test group. As a result, a test case might execute multiple times per test suite run — one time in each test group in which it is a member. Uncheck the box (default) to ensure that a particular test case can be a member only of the first test group for which it passes a filter. As a result the test case will execute only once per test suite run. **Note** All test cases starting with "**_**" (the **_setup** and **_cleanup** test cases) are excluded from the test case filtering process.

3 You will now add as many test groups as are needed to define the test suite. Typically, you define a test group for each set of criteria that determine whether to execute or not execute a particular test case when the test suite runs. That is, you will define filters that determine which test cases from all specified folders should run.

To create a new test group and add it to the test suite, click **Add**  . On the **Test Group** dialog box, supply a name for the test group and click **OK**. iTest adds the group to the **Test Groups** list.

4 In the **Test Groups** list, double-click a group to configure its properties. We discuss adding and configuring a test group in “Configuring a test group: The Test Group page” on page 405.

Managing the list of test groups for a test suite

Use the buttons on the **General** page to manage the list of test groups for the current test suite.

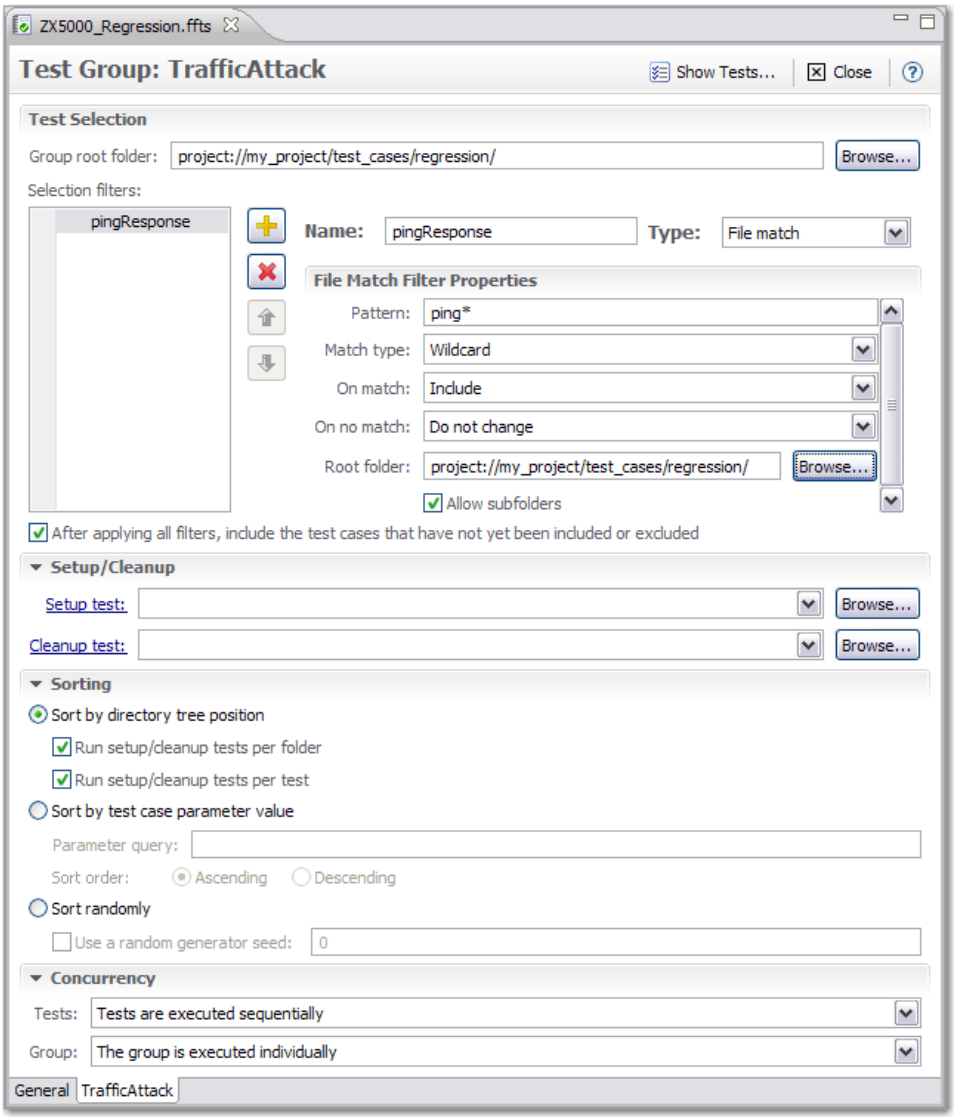
When you add a test group to the list, iTest adds it after the currently selected group. Test groups are executed in the listed order.

Add	Add a new test group to the list. The new group is added to the list after the selected group.
Rename	Rename the test group.
Remove	Delete the selected test group.
Move Up **Move Down**	Move the selected test group one up or down in the list. Test groups execute in the listed order.

Configuring a test group: The Test Group page

Test cases that are members of a test group are the test cases that will execute when the group runs (remember that test groups run as part of the test suite that contains the group).

Use the **Test Group** page to define filters that determine whether tests are included in the group or are excluded from it. Examples: Include test cases whose names start with the text “stress”, exclude test cases in a particular folder, or include tests that are owned by a particular person.



Editing a test group

1. You got to this page by double-clicking the test group in the **Test Groups** list on the **General** page of the Test Suite editor.
2. Specify the **Group root folder** — the folder that includes the test cases that should be in the test group (depending on the filters you design, some test cases in the folder will not be included in the group).

Note While creating the test suite document using the wizard, if you checked the **Filter files based upon matching criteria** option, then this value is filled in.

3 In the **Selection Filters** section, click **Add** to add a filter and then specify the following properties for the new filter:

Name	Specify a meaningful name for the filter. For example, **deleteLogMsgs** or **includeOnlyPortStatus**
Type	While you created the test suite document using the wizard, if you checked the **Filter files based upon matching criteria** option, then the **Type** is set to **File match**, and you can continue at Step 4. Specify how to interpret the pattern that you specify for the **Pattern** property: • **File match** (default): The **Pattern** is compared with the names of the files in the folder. Match with the filename. You can use the * wildcard character. **Note** All test cases starting with "_" (the _**setup** and _**cleanup** test cases) are excluded from the test case filtering process. • **Description match**: Match text in the **Description** property string for the test case. • **Owner match**: Match text in the **Owner** property string for the test case (typically, a person's name). • **Parameter match**: Specify a parameter setting Include or exclude test cases with a matching parameter setting. (for example, **testType=stress**). • **Parameter assertion**: Specify a **<paramName = value>** assertion. Include or exclude test cases for which the assertion is true.

File Match Filter properties

4 The properties that appear in this section of the page depend on the **Type** of match that you specified. Follow the directions appropriate to the **Type** that you specified.

Tip In some cases, you might design the filters so that, after filtering is complete, you also want to include tests that lie outside the filters. In such a case, you can check **After applying all filters, include the test cases that have not yet been included or excluded.**

Type: File match

While you created the test suite document using the wizard, if you checked the **Filter files based upon matching criteria** option, then the values in this table are already filled in.

Note All test cases starting with "_" (the **_setup** and **_cleanup** test cases) are excluded from the test case filtering process.

Pattern	Type the text that should be used as the basis for filtering test cases. For example, you might specify a **Pattern** of ***regression*** to add only tests with the text "regression" in the filename.
Match type	**Wildcard**: The filename must match the text specified for the **Pattern** property. **Regular Expression**: Interpret the text specified for the **Pattern** property as a regular expression when comparing the filename to the **Pattern**. **Strict**: The filename must exactly match the text specified for the **Pattern** property.
On match	Take the specified action when a test case's filename matches the pattern: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On no match	Take the specified action when a test case's filename does not match the pattern: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
Root folder	Browse to the folder that contains the tests that the filter should be applied to.
Allow subfolders	Check the box to search subfolders when applying filters. Default: Checked

Type: Description match

Pattern	Type the text that should be used as the basis for filtering test cases. For example, you might specify a **Pattern** of ***regression* (with** a **Match type** of **Wildcard**) to add only tests with the text "regression" in the **Description** property text string.
Match type	**Wildcard**: The **Description** property text string must match the text specified for the **Pattern** property. **Regular Expression**: Interpret the text specified for the **Pattern** property as a regular expression when comparing the **Description** property text string to the **Pattern**. **Strict**: The **Description** property text string must exactly match the text specified for the **Pattern** property.

On match	Take the specified action when a test case's **Description** property text string matches the pattern: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On no match	Take the specified action when a test case's **Description** property text string does not match the pattern: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.

Type: Owner match

Pattern	Type the text that should be used as the basis for filtering test cases. For example, you might specify a **Pattern** of ***Yujie*** (with a **Match type** of **Wildcard**) to add only tests with the text “Yujie” in the **Owner** property text string.
Match type	**Wildcard**: The **Description** property text string must match the text specified for the **Pattern** property. **Regular Expression**: Interpret the text specified for the **Pattern** property as a regular expression when comparing the **Owner** property text string to the **Pattern**. **Strict**: The **Owner** property text string must exactly match the text specified for the **Pattern** property.
On match	Take the specified action when a test case’s **Owner** property text string matches the pattern: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On no match	Take the specified action when a test case’s **Owner** property text string does not match the pattern: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.

Type: Parameter match

Pattern	Type the text that should be used as the basis for filtering test cases. For example, you might specify a **Pattern** of ***stress*** (with a **Match type** of **Wildcard**) to add only tests with the parameter value of “stress”. You specify the particular parameter to test with the **Query** property.
Match type	**Wildcard**: The filename must match the text specified for the **Pattern** property. **Regular Expression**: Interpret the text specified for the **Pattern** property as a regular expression when comparing the filename to the **Pattern**. **Strict**: The filename must exactly match the text specified for the **Pattern** property.
On match	Take the specified action when the filter returns **True**: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.

On no match	Take the specified action when the filter returns **False**: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
Query	Specify the query path to the parameter (its path in the parameter tree).
On no parameter	Take the specified action when a test case does not include the parameter specified for the **Query** property: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.

Type: Parameter assertion

Query	Specify the query path to the parameter (its path in the parameter tree). The query returns the value in the **$value** variable for testing in the assertion that you specify in the **Assertion** property.
Assertion	Based on the value returned by the query that you specified in the **Query** property, specify an expression to evaluate. For example, the assertion **$value == 42** tests whether the value returned by the query is equal to 42.
On no parameter	Take the specified action when a test case does not include the parameter specified for the **Query** property: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On assertion true	Take the specified action when the filter returns **True**: **Include**: (default) Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.
On assertion false	Take the specified action when the filter returns **False**: **Include**: Include the matching test in the group. **Exclude**: Exclude the matching test from the group. **Do not change**: (default) Do not include or exclude the test case. Instead, consider the test again when applying the next filter in the list of filters.

5 As mentioned earlier: In some cases, you might design the filters so that, after filtering is complete, you also want to include tests that lie outside the filters. In such a case, check **After applying all filters, include the test cases that have not yet been included or excluded**.

6 Now, you will specify whether to execute setup and cleanup test cases.

Global Setup / Cleanup properties

Global setup test	Specify the test case to execute before executing the test cases in the test group. The test case typically brings all devices and applications in the testbed to a known state before testing starts.
Global cleanup test	Specify the test case to execute after executing the test cases in the test group. The test case typically brings all devices and applications in the testbed to a known state after testing finishes.

Sorting properties: Specify the order of execution

Specify one of the following methods of determining the order in which the test cases should execute:

Sort by directory tree position		Order the test cases for execution in the order they are listed in the root folder. If you specify this sorting option, then you can further specify whether to run setup or cleanup test cases, as follows:
These options are applied only when you select **Sort by directory tree position**	**Run setup/cleanup tests per folder**	Check the box to execute the **_setup** and **_cleanup** test cases for each folder in the root folder that includes test cases that are included in the test group. See “Configuring setup and cleanup test cases for folders” on page 401. Default: Checked
	Run setup/cleanup tests per test	Check the box to execute the **_setup_each** and **_cleanup_each** test cases for each test case that is included in the test group. See “Configuring setup and cleanup test cases for folders” on page 401. Default: Checked
Sort by test case parameter value		Order the test cases for execution in the order of the value returned for the specified parameter.
	Parameter query	Specify a query that returns the value of the parameter used to sort
	Sort order	**Ascending**: (default) Sort alphabetically increasing sort by parameter value. **Descending**: Sort alphabetically increasing sort by parameter value.
Sort randomly		Order the test cases for execution in random order.
	Use a random generator seed	Use a particular number to start the process of determining the random sort order.

Concurrency

Tests	Specify whether tests should execute in parallel with other running tests (concurrently) or each test should wait for the preceding test to complete before executing (sequentially). Tests are executed concurrently Tests are executed sequentially (default)
Group	Specify whether this group should run to completion (all tests finish executing) before the next group runs (individually) or this group should run in parallel with other running groups (concurrently). The group is executed individually The group can be executed concurrently with others

Renaming a test group

1. On the **General** page of the Test Suite editor, select the test group in the list.
2. Click **Rename** .
3. Type a new name for the test group and then click **OK**.

Reviewing the tests that are members of a test suite or of a test group

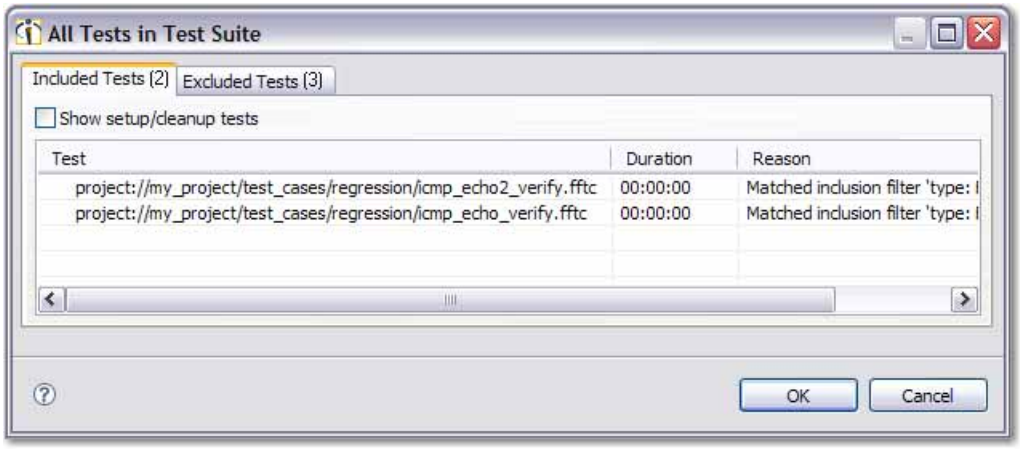
The test cases that are members of a test group are the ones that will execute when the group runs. Membership is determined by applying the **include**/**exclude** properties to all of the test cases in the root folder. (You set the properties in the **File Match Filter** properties section of the Test Suite editor's test group page.) All test cases in a test suite's groups are member of the suite.

Viewing the test cases in a test suite or test group

Viewing the included and excluded test cases in a test suite

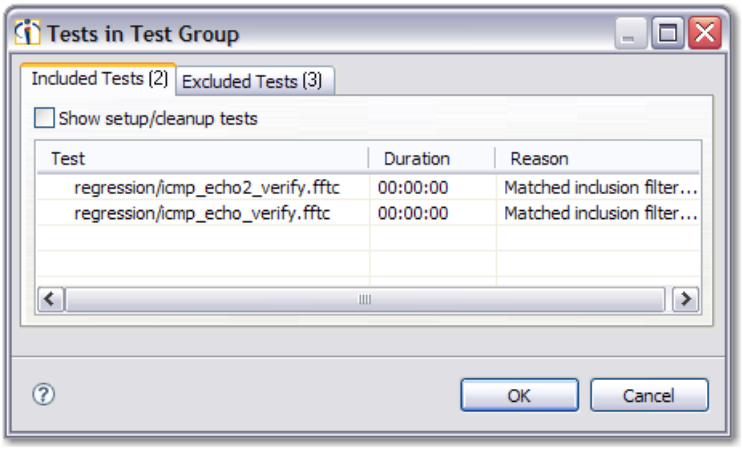
On the Test Suite editor General page, click **Show All Tests** in the toolbar. The **All Tests in Test Suite** dialog box displays the list of all tests that will execute when the test suite runs.

Note The tab title includes the count of tests that appear on the page.



Viewing the included and excluded test cases in a test group

On the **Test Group** page in the Test Suite editor, click **Show Tests**. The **Tests in Test Group** dialog box displays the list of all tests that will execute when the test group runs.



- On the **Included Tests** tab, member test cases appear in the order in which they will be executed. The order is determined by the **Sorting** setting that you made on the test group's page in the Test Suite editor (as described in "Sorting properties: Specify the order of execution" on page 412).
- To display the setup and cleanup test cases in the list (also in execution order), check the **Show setup/cleanup tests** check box. You can specify two levels of setup/cleanup:
  - Setup/cleanup before/after executing the test cases in a folder and/or before/after executing each test case. See "Configuring setup and cleanup test cases for folders" on page 401.
  - **Global setup** and **Global cleanup** before/after executing all of the test cases in a group. See "Global Setup / Cleanup properties" on page 412).
- The **Excluded Tests** tab lists the tests in the root folder that were excluded from the group for the reason displayed in the **Reason** cell.

Columns on the page

Click a column header to sort by the column.

Test	The test group name appears at the top level and test cases at the next level. While viewing the list for a test suite, t the path for a test case is relative to the workspace root. While viewing the list for a test group, the path for a test case is relative to the root folder for the group.
Duration	This cell displays the value of the **Estimated execution time** property for the test case as specified on the Test Case editor **General** page.
Reason	The text explains why a given test is included or excluded. A test cases can be excluded, for example, because it did not match an inclusion filter, because it matched an exclusion filter, or because it is already included in group and you had not checked the **Tests can be in multiple groups** property.

Running a test suite

To execute a test suite, you can do either of the following:

- Execute a test suite using iTestRT, as described in Chapter 32, “iTest Runtime: iTestRT”.
- Define an iTest job that references the test suite. When the job runs, the test suite executes. For information on defining, scheduling, and running jobs, see Chapter 18, “Scheduling Execution”.

Note itestcli does not support test suites.

CHAPTER 20

Debugging Test Cases

Note For ease of bug tracking and resolution, the manual steps performed during debug mode (e.g., breakpoints, manual pause) are captured in test reports. The steps captured during debug mode are highlighted in green in the test reports for ease of reference.

The following sessions support manual steps to be captured in test reports during debug mode: Chat, Command, Database, File, Flex, Http, Mail, Process, REST, SLS, SNMP, Serial, SSH, Swing, Syslog, Tcl Shell, Telnet, UDP, Web, Web Service, Wireshark, XML-RPC, VNC

Debugging: Executing procedures, Pausing, stopping, and single-stepping

In the **iTest Debugging** perspective or session windows that are opened by test cases, you can use tools to control execution.

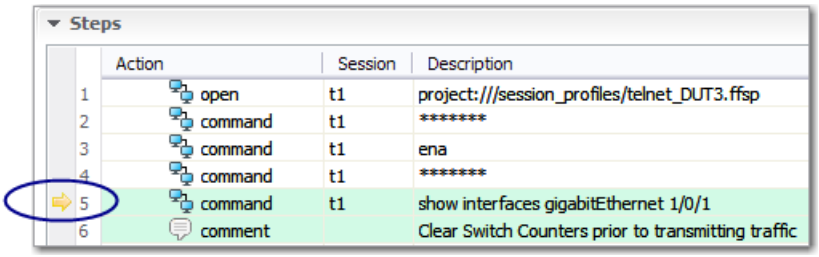
Stop, **Execute**, **Pause**, and **Load for Execution**

The **Execute One Step** and **Execute Procedure and Pause** buttons become active when the test is paused.

- **Execute One Step** is often called “step into”.
- **Execute Procedure and Pause** is often called “step over”.

While execution is paused, you can perform interactive (manual) actions in the session. iTest captures all test case steps and manual commands or QuickCalls that you perform. If a manual step is useful, you might later add it to the test case (or to any test case).

While execution is paused, the current step is marked in the Test Case editor with a yellow arrow.



Single-stepping and multiple threads

Single-step execution has unique behaviors when multiple threads are running. One thread is always designated as the “foreground thread,” and is highlighted in the Threads view. Each click of **Execute One Step** moves the procedure through the foreground thread one step at a time. For more details, see the Threads view.

Note When you click **Execute One Step**, the procedure pauses at the completion of each step in the foreground thread exactly as if it had reached a breakpoint. Other threads then pause on completion of their respective executing steps. If any other thread reaches a breakpoint, then that thread becomes the foreground thread from that point on.

Because of this behavior, We recommend that you use the Threads view to keep track of the foreground thread whenever you are single-stepping through a procedure with multiple threads.

Breakpoints and multiple threads

Remember that common situations such as asynchronous steps or calls to foreign procedures initiate new threads. If a procedure is running multiple threads, then when any thread reaches a breakpoint, all threads pause upon completion of the executing step in each thread.

For more details on thread behavior, see the Threads view.

Note iTest ignores breakpoints on skipped steps — the steps are skipped without pausing execution.

Stopping execution

Click **Stop** .

iTest allows the currently executing step to complete execution and then stops executing.

The Execution view displays an execution message indicating that execution was canceled by the user and another message that identifies the test case and states that it aborted.

Pausing and resuming execution

Click **Pause** .

The window switches to **iTest Debugging** perspective.

The Test Case editor opens and identifies the current step with a yellow arrow in the left column.

The currently executing step completes execution and then execution pauses.

The next step does not start until you click **Start Execution** , **Execute One Step** , or **Execute Procedure and Pause** (often called **“**step over**”**).

When you resume execution, the window switches back to the **Execution** perspective or session window and executes the next step.

Single-stepping through an entire test case

1 Load the test case using one of the following methods:

Before starting execution, in either the **Execution** perspective or the **iTest Debugging** perspective, click **Load for Execution** .

While working in the Test Case editor, save the test case and then click **Load for execution** .

In the iTest Explorer, right-click the test case and select **Load for execution** or **Load for execution – New Window**.

2 The Test Case editor opens to the first step in the test case. Execution is paused.

You can now change runtime parameter values in the Data view.

You can now perform interactive actions in the session to view the results. If a step is useful, you might later add it to the test case.

Use **Execute One Step** or **Execute Procedure and Pause** as needed.

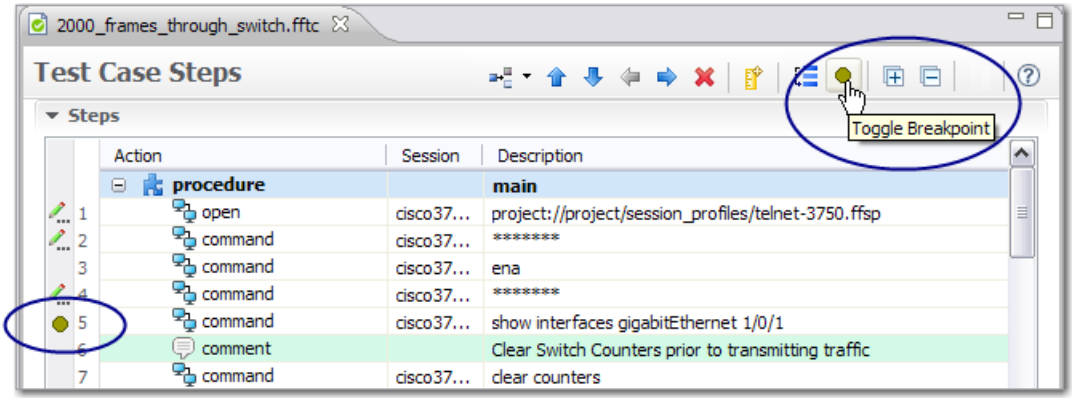
Single-stepping while execution is paused

While execution is paused, **Execute One Step** causes iTest to execute the very next step and then pause before executing the next step.

Breakpoints: Overview

Breakpoints tell iTest to pause execution.

In the Test Case editor, you apply a breakpoint to a selected step by clicking **Toggle Breakpoint** . An icon in the margin to the left of the step indicates the breakpoint.



Because breakpoints pause a test case until you manually resume execution, it is important to keep track of them. The Breakpoints view shows you every breakpoint in every test case in your workspace, all at once (even for test cases that are not currently open).

Adding a breakpoint

During execution, when iTest encounters a breakpoint, it switches to the **iTest Debugging** perspective to facilitate single-stepping and other debugging tasks.

Tip While execution is paused, you can perform interactive actions in the session to view the results. If a step is useful, you might later add it to the test case.

To add or remove a breakpoint

You can add or remove breakpoints at any time (even during execution or while execution is paused).

1. On the Test Case editor **Steps** page, select the step.
2. In the toolbar, click **Toggle Breakpoints** . A breakpoint is added or removed as appropriate.

For breakpointed steps, a marker appears in the first column.

Note iTest ignores breakpoints on skipped steps — the steps are skipped without pausing execution.

All current breakpoints are identified on the Breakpoints view.

Single-stepping through a test

Single-stepping is a very convenient way to closely monitor the operation of a test case. Using step-into or step-over in the main toolbar will ask iTest to execute one more step and then stop again. (Executing one step can take a long time in some cases but execution will stop when the next step is complete.)

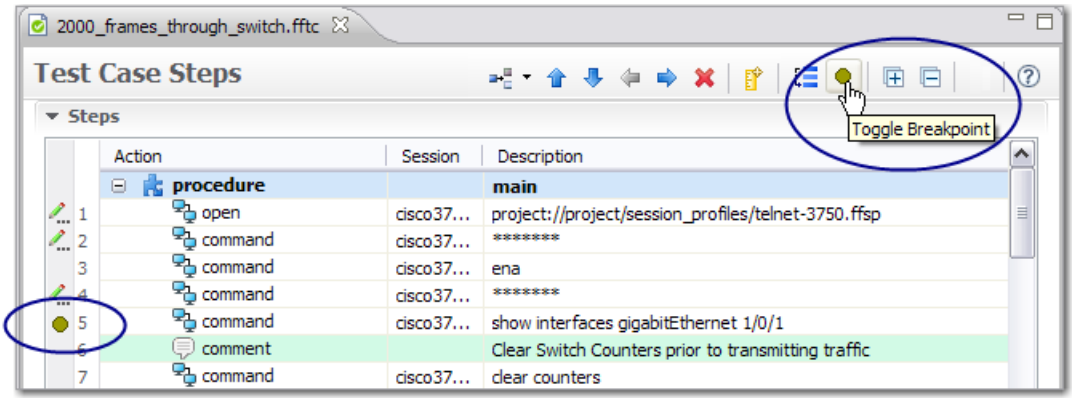
If multiple threads are running, you will be single-stepping in just one of the threads. When you start executing (even by single-stepping), all threads will start running again, and then execution will pause after the next step in the foreground thread is reached. But at that point, iTest will ask all other threads to pause after they complete the current step, and you will get control back again after all threads indicate that they have paused. If you would like to single-step through a different thread, you can select a different thread in the Threads view to make it the foreground thread for single-stepping.

Step-into is used to move to execute the next step. If the current step is a call step, then execution will stop after the first step in the called procedure. Step-over is identical except when the step is a call step, then execution will stop after the called procedure has completed execution and returned.

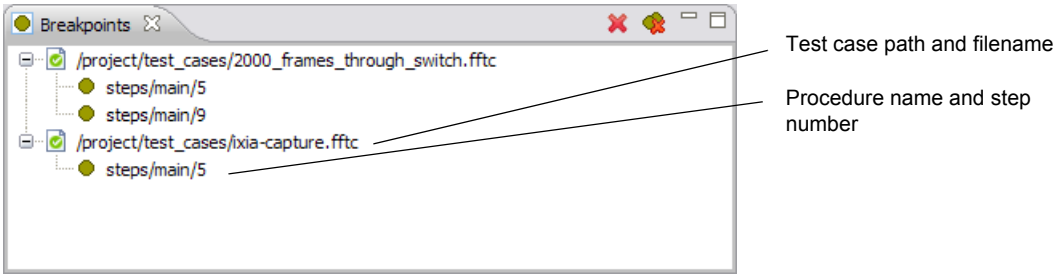
Breakpoints view

Breakpoints tell iTest to pause execution.

In the Test Case editor, you apply a breakpoint to a selected step by clicking **Toggle Breakpoint** . An icon in the margin to the left of the step indicates the breakpoint.



Because breakpoints pause a test case until you manually resume execution, it is important to keep track of them. The Breakpoints view shows you every breakpoint in every test case in your workspace, all at once (even for test cases that are not currently open). Here's an example:



Breakpoints view toolbar

	Delete the selected breakpoints
	Delete all breakpoints

During execution

During execution, iTest pauses ***before*** beginning execution of the breakpointed step. The breakpointed step is highlighted, and the test case is marked as paused in the Execution view.

Breakpoints and multiple threads

Remember that common situations such as asynchronous steps or calls to foreign procedures initiate new threads. If a procedure is running multiple threads, then when any thread reaches a breakpoint, all threads pause upon completion of the executing step in each thread.

For more details on thread behavior, see the Threads view.

Note iTest ignores breakpoints on skipped steps — the steps are skipped without pausing execution.

Resuming execution after a breakpoint

Single-step and multiple threads

Single-step execution has unique behavior when multiple threads are running. One thread is always designated as the “foreground thread,” and is highlighted in the Threads view. Each click of **Execute one step** moves the procedure through the foreground thread one step at a time.

Note When you click **Execute one step**, the procedure pauses at the completion of each step in the foreground thread exactly as if it had reached a breakpoint. Other threads then pause on completion of their respective executing steps. If ***any other*** thread reaches a breakpoint, then ***that thread becomes the foreground*** thread from that point on.

Because of this behavior, We recommend that you use the Threads view to keep track of the foreground thread whenever you are single-stepping through a procedure with multiple threads.

Adding and removing breakpoints

In the Steps section of the Test Case editor, select one or more steps and then right-click the step and select **Toggle Breakpoint**. Alternatively, select one or more steps and then click **Toggle Breakpoint** in the toolbar.

- **To remove a breakpoint while in the Breakpoints view**

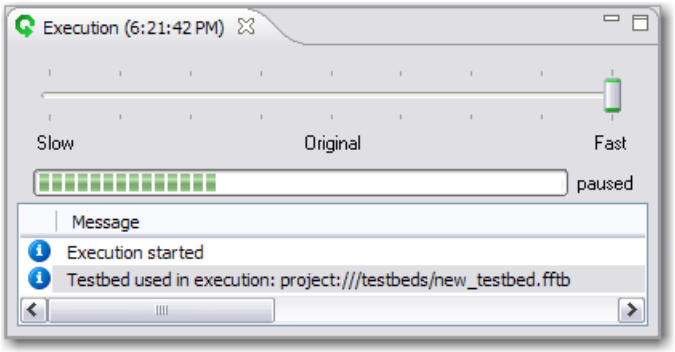
  You can remove a breakpoint from within the Breakpoints view without opening the affected test case.

  - To un-breakpoint steps: one or more steps and then click .
  - To remove every breakpoint in the Breakpoints view: click .

Adjusting execution speed

You can change execution speed during execution using the **Speed** control in the Execution view.

- During execution, change the execution speed at any time.
- Specify the default speed for any test case step in the **Timing > Start** properties group.



Fast: Submit commands as quickly as the computer can send them while allowing each step run to completion before starting the next step. Asynchronous (concurrent) steps, in contrast, are executed at original speed.

Original: Execute no faster than the original capture. Execution may be slightly slower because each step will start no sooner than the original delay from the previous step end, but may be later because it will wait for the previous step to complete before starting.

Slow: Use this setting to give you a chance to observe behavior that you might miss at a higher execution speed.

Activating the Speed control in the Execution view

The speed control does not appear in the Execution view by default. Follow these steps to display the tool.

1. Click **Window > Preferences**.
2. On the **Preferences** page, click **Spirent > Views > Execution View**.
3. Check **Display the execution speed slider in the Execution view.**

CHAPTER 21

Test Reports

Test reports

Whenever you execute a test case, Spirent iTest saves all of the steps and responses in a test report. Each test report records a single execution of a particular test case, so test reports are identified by the test case name and the time of execution. In addition, during debug mode (see "Debugging Test Cases" on page 417), all manual steps executed in the open sessions and their associated responses are captured and included in the test report.

Spirent iTest lists all available test reports in the Test Reports view and displays the details of a test report in the Test Report editor. There are many options for accessing test reports:

- From the Test Report editor or Test Reports view, you can save test reports into HTML, PDF, Text, XML, or XML_RAW formats.
- You can export test reports into a file suitable for import into other Spirent iTest workspaces.
- You can configure auto-export so that test reports are saved into a specified format whenever execution ends.

Tip Test reports are displayed after test cases finish execution. To view status for each step during execution, use the Console view (as described in "Console view" on page 295).

How test reports are stored

In conventional test automation solutions, test scripts often generate very large test report files. This can make it difficult to find the information you need in a report. For this reason, and performance reasons, Spirent iTest stores test reports in a database. While Spirent iTest executes a test case, it places execution information into records in an embedded database in your workspace (or optionally into a separate database that you specify). A test report, therefore, is not a specific file; rather it is a series of records in a database. The **.report** directory contains files that implement the default database.

Note The manual steps captured during debug mode are highlighted in green to indicate manual commands and separate them from the test scripts.

To make it easier for users to share report data, you can configure Spirent iTest to save test reports to an external database rather than to the default database within your workspace.

Uses for test reports

It is easy to export test reports as HTML, PDF, Text, XML, XML_RAW files for bug reports or for persons that do not use Spirent iTest (for descriptions of the various formats, see "Test report file formats" on page 449). Here are suggested uses of the reports:

- **Reproducing bugs**

  One of the most productive ways to use iTest is for bug tracking and resolution. Once a test demonstrates a bug, you can easily attach the test report, which includes the manual steps performed, to a bug tracking system and/or send it to a developer for review.

  All of the information that a developer needs to reproduce a bug appears in the single automatically-generated report — not only the exact steps (including the manual steps) required to reproduce the bug, but also references to the topology or testbed document that specifies exactly how to configure the testbed and the session profiles that specify the precise configuration settings for the devices or software applications used in the test case.

  In addition, the report includes the commands (multi-line commands) formatted with appropirate line breaks (as entered during test development), which makes it easy to parse. See "Example HTML test report as viewed on a browser" on page 428.

  Note For ease of bug tracking and resolution, the manual steps performed during debug mode are also captured in test reports and highlighted in green to indicate manual commands.

  iTest enters debug mode when you pause execution of test case (breakpoints, manual pause). The steps captured during this mode is highlighted in green in test reports for ease of reference. See “Debugging Test Cases” on page 417.

- **Comparing the behavior of two versions of the same software**

  When you receive a new release of software, you might want to run a “sanity check” test case that identifies any undocumented changes in the behavior of the software that will affect your existing test cases. The sanity check might execute some basic **show version** and **show configuration** commands that return important system and configuration information. Spirent iTest makes it easy to compare the test report for the “new” sanity check report (for example, against version 4.8) with the “reference” sanity check report (against version 4.7) so you can quickly identify changes in behavior between the two versions of software. You can view the reports side-by-side with highlighted “diffs”. See "Comparing (“diffing”) two test reports" on page 444.

- **Comparing a failed run with the last passing run**

  When a test run fails, your first debugging task is typically to compare the failure with the last known passing run. Spirent iTest makes it easy to compare the two test reports to pinpoint the location of the failure. You can view the reports side-by-side with highlighted “diffs”. See "To compare two reports" on page 444.

- **Generating response maps**

  Test reports saved in HTML, PDF, Text, XML, XML_RAW format include verbatim response data. The automation engineer can use the test reports generated by a manual tester to create response maps and add analysis to test cases — all without having to rerun a test case or set up a topology or testbed.

  Tip You have the option to save only selected sections of any report when you export it. "Customizing the content of a test report" on page 459.

- **Generate a report on measurements taken during the test execution**

  You may set a test case to extract data and report the extracted data in your custom HTML reports to analyze the measurements taken during the test execution.

  See "Execution Message processor" on page 659 in Chapter 29, “Analysis Rules: Validating Responses”.

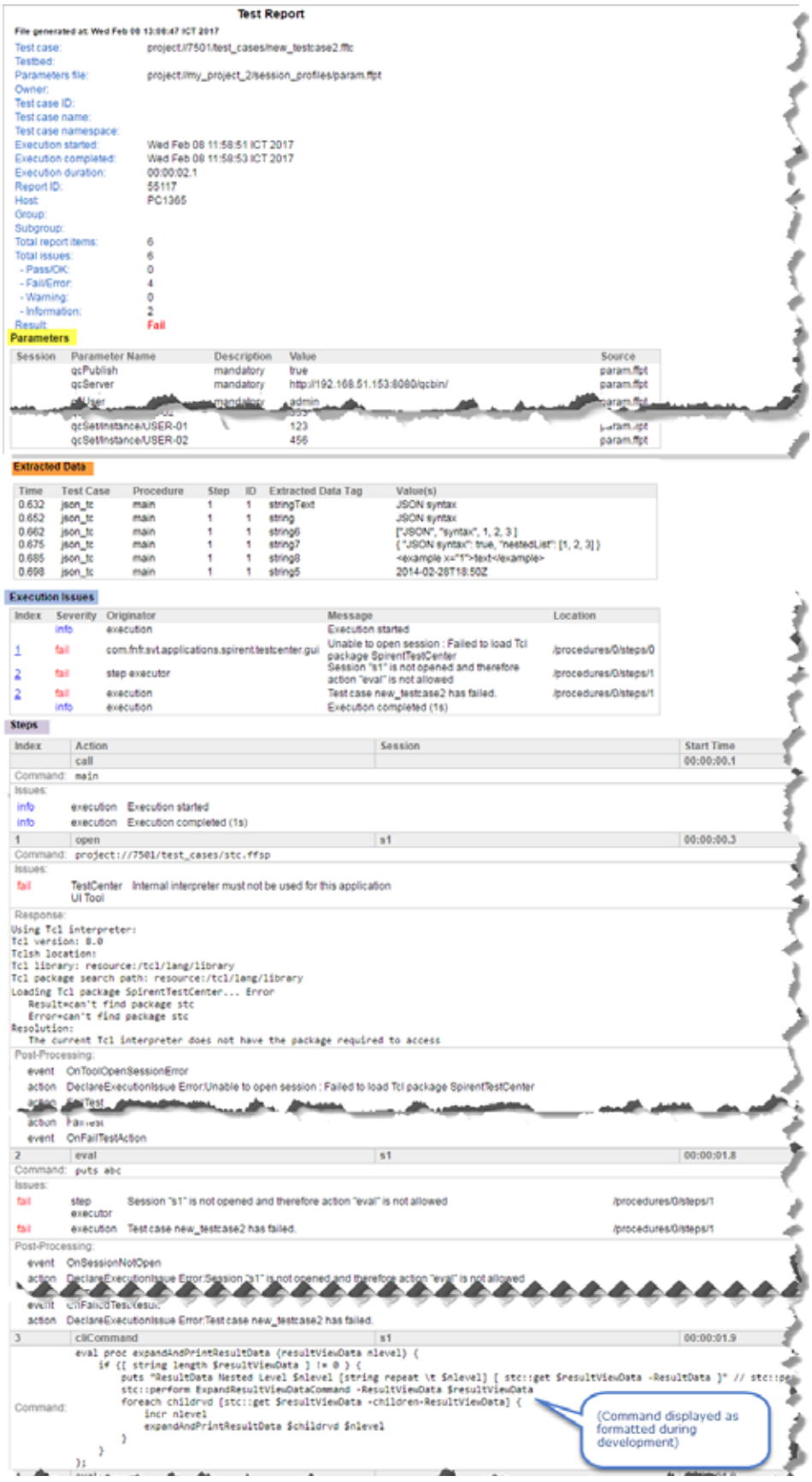
  See also "Example HTML test report as viewed on a browser" on page 428 for a sample report with showing the extracted data.

  Limitations

  The following limitations apply for the data extracted for each execution:

  - **Total elements stored:** A maximim of 128 extracted data items per execution.
  - **Bytes stored:** A maximim of 128 characters of any tag or value. Any tag or value that exceeds 128 characters will be truncated.
  - **Array elements stored:** Any extracted data item whose value is an array that exceeds 128 items will be rejected (discarded).

Example HTML test report as viewed on a browser



Screen snapshots in HTML-format reports

Test reports for session types that use browsers (Swing and Web) can include thumbnail images of screen snapshots that are taken during execution. Click a thumbnail to view the full-size image.

Screen snapshot in HTML Report—Response in JSON format reports

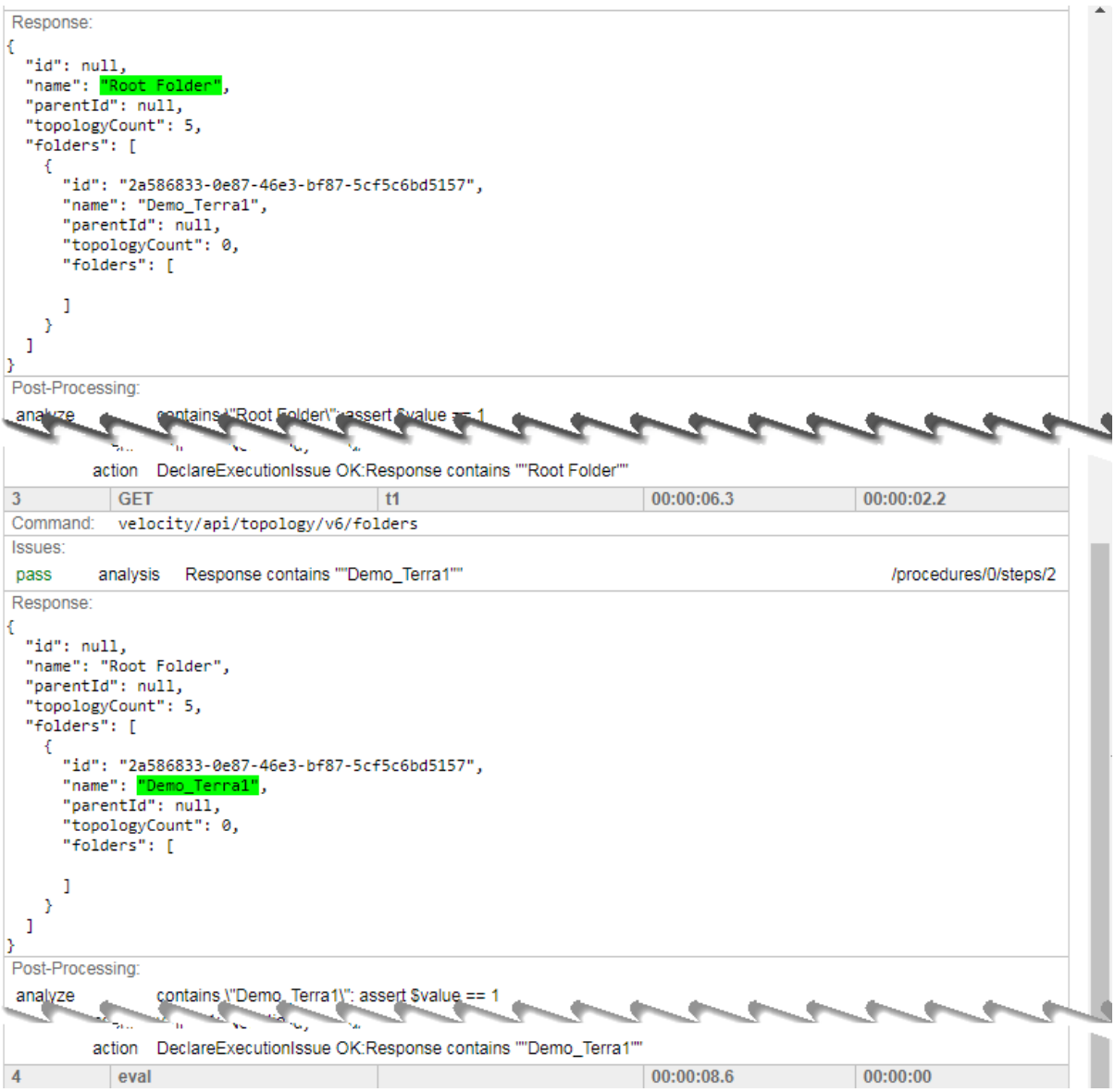
On the test report's **Response** section, iTest displays response format as **JSON** or **Text** form (as auto-detected).

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.

  Only text indentation is applied (to HTML, HTML/JSON, PDF, and customized reports) and not the text color.

- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").



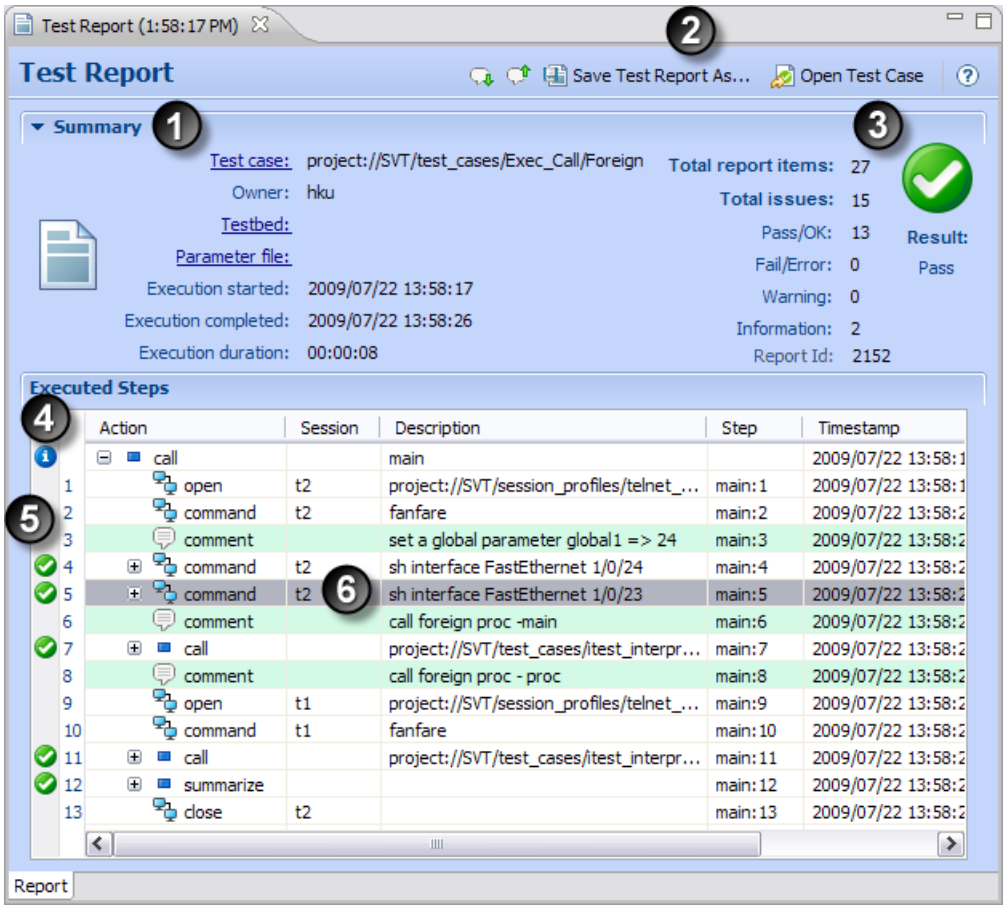
Test Report editor

By default, the Test Report editor displays the report as soon as execution ends. For older test reports, double-click the report in the Test Reports view to open the report in the Test Report editor.

In the report, the steps are organized into a hierarchy matching the procedures and steps that were executed. To make it easier for you to troubleshoot execution, the report looks very much like the **Test case** editor: a grid of steps and analysis rules with events occurring during execution nested under step row.

- While working on a test case in the Test Case editor, the fastest way to view a recent test report is to click in the toolbar. Click  to open the most recent report in the Test Report editor. Click the arrow to display the list of the five most recent reports and then select a report to open it.
- The Test Report editor displays only executed steps, skipped steps are not included.
- You can click and to open and close procedures, QuickCalls, analysis rules, and nested step constructs in the report.
- To set a preference for displaying or not displaying test reports when execution ends, see "Setting preferences for execution" on page 281.
- You can specify a preference for how long to store test reports in Spirent iTest's built-in database (that is, when to start discarding old reports). See "Setting preferences for Test Reports" on page 455.
- To make it easy to distinguish between a test report and its associated test case, the background color for test reports is blue by default. For instructions on configuring a different background color, see "Setting preferences for Test Reports" on page 455.
- To ensure best execution speed, Spirent iTest saves test report data only while the session is not actively communicating. As a result, there may be a delay in displaying the report after execution ends. You can specify the maximum wait. See "Setting preferences for Test Reports" on page 455.

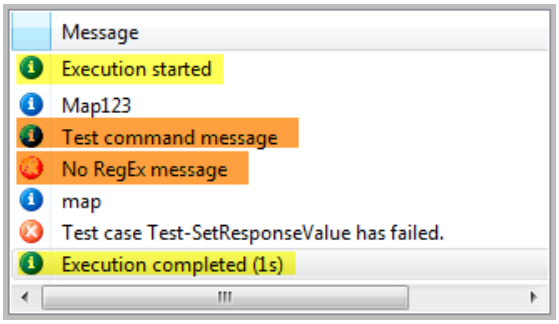
Viewing test reports



1 Summary information on the executed test case

This section (collapsible by clicking the arrow Summary) displays the URI of the test case, the URI of the topology or local or global testbed used for execution, time and duration of execution, counts of execution issues, final test result, and a **Report ID** that uniquely identifies the report.

2 Toolbar:

Go To Next Issue **Go To Previous Issue**	Test reports display icons that represent **execution issues** that are generated during execution. Each issue can have an associated **execution message**. Hold the cursor over the icon to read the execution message that is associated with the execution issue. Execution issues and their associated execution messages also appear in the Step Issues view and Execution view. Each message has a **Severity**: Information OK (this severity is listed as "pass" in test reports that have been saved in HTML, PDF, Text, XML, and XML_RAW formats) Warning Error (this severity is listed as "fail" in test reports that have been saved in HTML, PDF, Text, XML, and XML_RAW formats)  Some execution issues are built-in (for example, Execution started, Executions completed). Others are defined by the test case developer (for example, "Test command message"—see "The 'message' action: Add a severity type and message type" on page 247 in Chapter 11, "Actions"). For very long test reports, it might take some time to locate execution issues. The **Go To Next Issue** and **Go To Previous Issue** buttons make it easier to quickly locate and view issues and their associated execution messages.
Save Test Report As...	Export the test report as described in "Uses for test reports" on page 426.
Open Test Case	Select a step in the report and then click the button to view the associated step in the Test Case editor. If no step is selected, then the Test Case editor opens the test case. Alternatively, right-click a step in the report and select **Open Test Case**.

3 Test result

An icon represents the overall test result: **Pass**, **Fail**, **Abort**, or **Indeterminate.** The **Indeterminate** result means that, because no pass/fail rule was executed, Spirent iTest cannot set the result as either **Pass** or **Fail.**

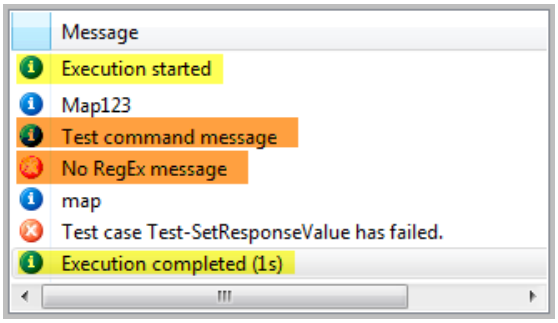
4 Columns in the report

[first column]	**Step index** numbers appear in the first column. Notice that, due to changes to execution flow (for example, due to a loop), the numbers may differ from the **Step ID** numbers in the test case. You can set a preference to display or hide the index numbers. See "Setting preferences for Test Reports" on page 455.
Action	Specifies the action executed in the step.
Session	Identifies the session in which the step was executed. In procedures with more than one session, this will help you match the step to its session.
Description	The **Description** column identifies the action, analysis rule, or event and provides detailed information about the executed step, such as the command executed or the session profile used for an **open** step.
Step	Identifies the executed step by procedure name and step ID number using **procedureName:stepNumber** format.
Timestamp	Date and time of execution for the step.
Duration	Time for completion (including analysis) for the step.
Thread	Identifies the thread in which the step executed.

5 Execution issues

Hold the cursor over the execution issue icons to view the execution message that is associated with the execution issue. The icon reflects the severity of the execution issue: **Information**, **OK**, **Warning**, and **Error**. The messages also appear in the Execution view and the Step Issues view.

Some execution issues are built-in (for example, Execution started, Executions completed). Others are defined by the test case developer (for example, “Test command message”—see "The ‘message’ action: Add a severity type and message type" on page 247 in Chapter 11, “Actions”).



Note **OK** severity is listed as “pass” in test reports that have been saved in HTML, PDF, Text, XML, and XML_RAW formats. **Error** is listed as “fail”.

6 Selecting a row in the report

If the **Response view** is open, select a step to display its response. If the Response view is not open, double-click a step to view the response in the view.

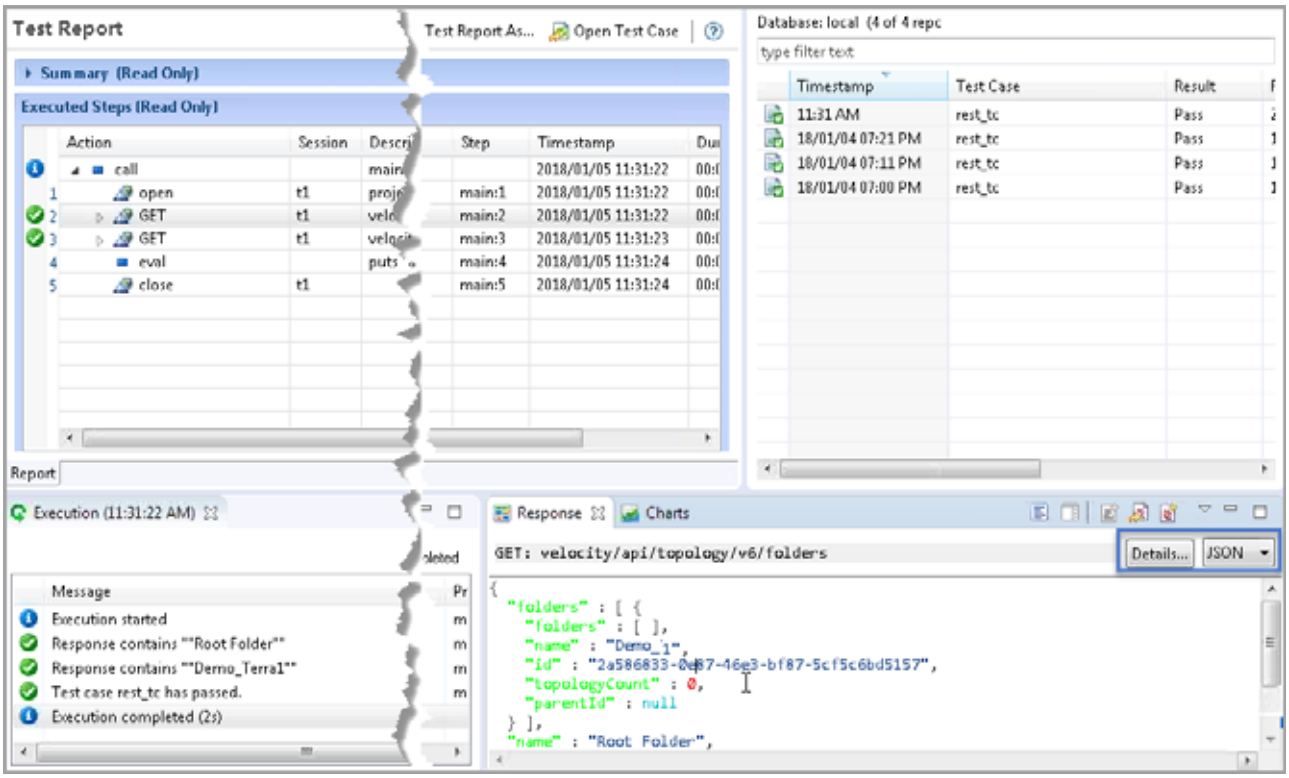
On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form.

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.

- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, “JSON Editor”).



If a step used a field replacement, then the resulting substituted text appears in the report.

For **run** steps (that run child test cases), you can open the test report for the child test case: Right-click the **run** step and select **Open Test Report**.

Note The test report of a master test case is also generated and includes reports of each individual child/slave test cases within it.

Whereas, reports generated for a test suite includes individual test case report and not the overall test suite report (unlike master test case report).

Test Reports view

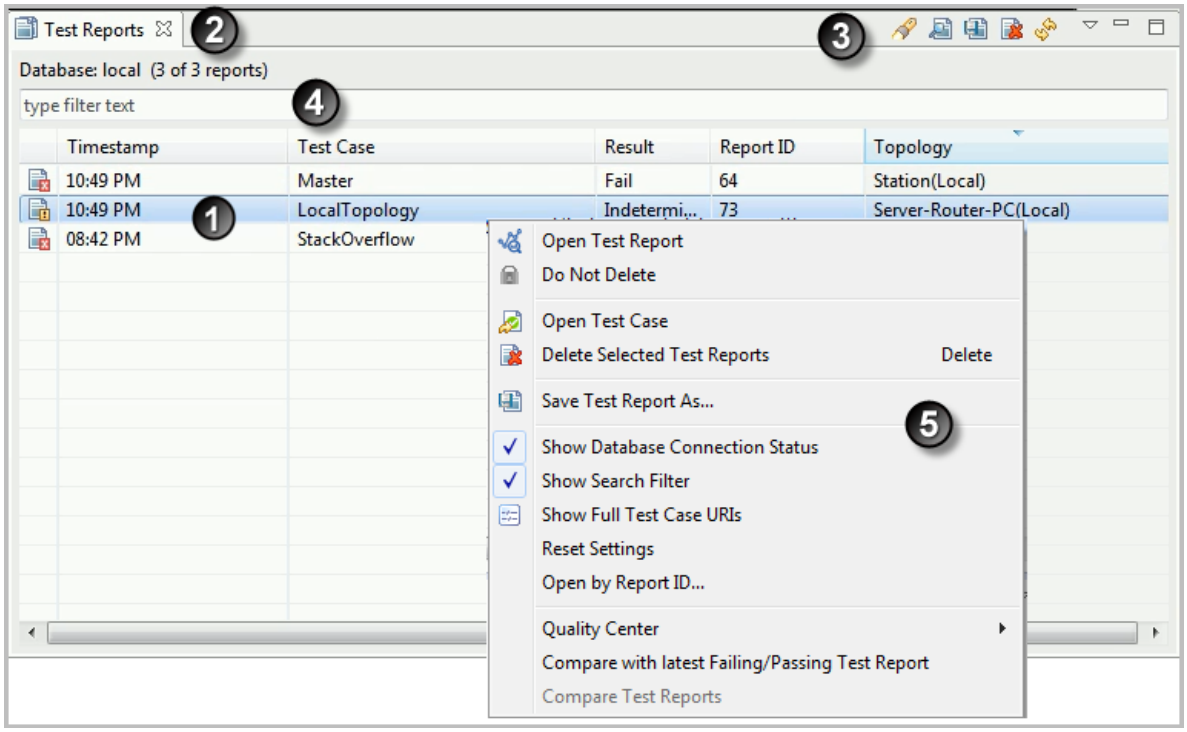
Whenever you execute a test case, Spirent iTest saves all of the steps and responses as a test report. Each test report relates to a single execution of a specific test case, so test reports are identified by the test case name and the timestamp of execution.

In contrast with the Test Report editor (which displays the actual report content), the Test Reports view displays a list of test reports in the database. By default, reports are ordered by execution time, but you can specify other sortings by clicking a table heading.

To open the Test Reports view

- While working in the Activities perspective, click **Browse test reports**.
- While working in any other perspective, click the arrow on Show View and then select **Test Reports**.

Working in the Test Reports view



1 List of test reports

- In the example, we see that four reports are listed (one per line). You can control which reports appear in the list using two tools: a database search, and a filter that you can apply to the results of the search. Both tools will be described in a moment.
- The icon in the first column represents the execution **Result** of the test case (**Pass**, **Fail**, **Abort**, or **Indeterminate**)
- Double-click a report to view it in the Test Report editor (described in "Test Report editor" on page 431).
- Right-click a report to view a menu of options, as described in 5 "Context menu (right-click) options" on page 438.
- For Spirent iTest's built-in database, you can specify a preference for how long to store test reports (that is, when to start discarding old reports). See "Setting preferences for Test Reports" on page 455.

Sorting reports in the view

By default, Spirent iTest lists reports in chronological order. You can click any heading to sort on a property of the report.

- Click the **Timestamp** heading to sort in chronological or reverse chronological order.
- Click the **Test Case** heading to sort in alphabetical order or reverse alphabetical order. Within each group of identically-named test cases, reports are sorted by **Timestamp**.
- Click the **Result** heading (the first column) to sort in the following order: **Pass**, **Indeterminate**, **Fail**, and **Abort**. Within each **Result** group, reports are sorted by **Timestamp**.
- Click the **Report ID** heading to sort in numerical order or reverse numerical order.
- Click **Topology** heading to sort in alphabetical order or reverse alphabetical order. Within each group of identically-named topology names, reports are sorted by **Timestamp**.

Timestamp	Date and time that the test case was executed (tests executed today display only the time).
Test Case	Name of the test case that executed to generate the report.
Result	Test execution result: **Pass**  , **Fail**  , **Abort**  , or **Indeterminate** 
Report ID	Indicates the ID of the report.
Topology	The name of the topoly used in the test case. For example, if you have a master_suite that is running on different topologies, in the test reports view, you will see the master_suite. You may click on each of the test case reports to see the topology used.
Host	If you have configured iTest to save test reports to an external database, then this column appears and displays the hostname of the iTest client that generated the report.
Group	If you have configured iTest to save test reports to an external database, then this column appears and displays the group name that was configured on the iTest client.
Subgroup	If you have configured iTest to save test reports to an external database, then this column appears and displays the subgroup name that was configured on the iTest client.

2 Source of the test reports and connection status

This line displays whether the test reports originate in the **Internal** (built-in) database in your current workspace (the default source) or in an external database. For more information, see "Configuring Spirent iTest to save test reports to an external database" on page 447.

If there is an issue with the database connection, then the error message appears here.

3 Test Reports view toolbar

	**Search Database** Opens a dialog box that enables you to configure a database search that populates the list of reports in the view (that is, reports that meet the search criteria are listed). See "Controlling which reports appear in the Test Reports view: Database search and filter" on page 440.
	**Open the selected test report** in the Test Report editor (described in "Test Report editor" on page 431).
	**Save the selected test report as HTML, PDF, Text, XML, or XML_RAW**: See "Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file" on page 451.
	**Delete** the selected reports (Use Ctrl+click and Shift+click for multi-select.) Alternatively, press the **Delete** key.
	**Refresh** the view. Useful when new reports might appear due to ongoing execution.
	**Displays a list of options**. Described in "Context menu (right-click) options" on page 438

4 Filter

The  button allows you to populate the view with the results of a database search (that is, only reports that meet the search criteria are listed). You can further limit the reports that appear in the view by applying a filter to the results of the search.

When you type text into the **Filter** text box at the top, Spirent iTest applies the filter to the current list and then displays only reports that “pass the filter” (the associated test case name includes the specified text). To reset so that no filter is applied, delete the text. See also "Controlling which reports appear in the Test Reports view: Database search and filter" on page 440.

5 Context menu (right-click) options

Right-click a report to view a menu of options:

 **Open Test Report**	Open the selected report in the Test Report editor (described in "Test Report editor" on page 431).
 **Do Not Delete**	Lock the selected reports so they are not deleted from the list even when you click  **Delete Selected Test Reports**. Use Ctrl+click and Shift+click for multi-select
 **Open Test Case**	Open the test case associated with the selected report.
 **Delete Selected Test Reports**	Use Ctrl+click and Shift+click for multi-select Alternatively, press the Delete key.
 **Save Test Report As**	**Save the test report as HTML, PDF, Text, XML, or XML_RAW**: See "Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file" on page 451.
Show Database Connection Status	Alternate between displaying and not displaying the database server name and location (or connection error state) in the first line of the Test Reports view.

Show Filter	Show or hide the **Filter** text box at the top of the view.
Show Full Test Case URIs	Alternate between displaying test case names and test case URIs in the **Test Case** column.
Reset Settings	Reset all Test Reports view settings to the defaults.
Compare with latest passsing test report	Compare the selected report with the latest passing test report. This oprion is not available when you select more than one report.
Compare test report	Select two test reports and then click Compare test reports. iTest generates a comparison report for your reference. This option is not avaialble when you select only one report or more than two reports.
Quality Center	This option appears only if yo u use iTest with Quality Center. To start the **Publish Test Report to Quality Center** wizard, select one or more reports, right-click and select **Quality Center > Publish Test Reports to Quality Center.**

Controlling which executed steps appear in test reports

By default, each step that executes in a test is added to the test report. You have the option to specify that a particular step and any of its children should not appear in test reports (the step executes normally, it simply is not reported).

Tip You can improve test case performance by not reporting common framework function steps and procedure **call** steps.

Execution issues

If a child step of any step that is configured not to appear in reports has an execution issue, then, in the list of execution messages in the Execution view, the issue's icon appears next to the message for its nearest ancestor. The step index for the issue is associated with the child step that had the issue, so you can double-click the issue to open the Test Case editor to the child step that had the issue.

- **To specify that a step should not appear in test reports**

  1 In the Test Report editor, select the step or steps.

  2 In the **General** properties group, uncheck **Include this step and its children in test reports**.

- **To override the "no-report" setting: To specify that all steps should appear in test reports, regardless of the settings for the individual steps**

  You might want to include all steps in test reports, even though individual steps might be configured not to appear. Follow this procedure:

  1 Click **Window > Preferences**.

  2 On the **Preferences** page, in the **Spirent** group, navigate to **General > Execution**.

  3 Check **Include all steps in test reports (ignore the setting for the step)**.

- **To override the "no-report" setting in iTestRT and itestcli**

  iTestRT and itestcli use Boolean arguments to specify that all steps should appear in test reports, regardless of the settings for the individual steps:

  iTestRT: **--reportallsteps**

  Itestcli: **-ra** (**--reportAllSteps**)

Including system timestamp for individual steps in HTML test reports

By default for each step, HTML-format test reports include the time elapsed since test case execution started. To display, instead, the system timestamp for each executed step, you must edit the **item.xslt** file to uncomment one section and comment another. Comments begin with **<!--** and end with **-->**. To uncomment a section, delete both sets of characters.

File path: **project://resources/reportsXX/test_report_templates/HTML/item.xslt**

Sections to comment or uncomment as needed:

```
<!-- Uncomment to print time elapsed since test case execution started
~~~~~~~~~~~~~~~~--> (Uncommented by default)
```

or

```
<!-- Uncomment to print system time the individual step began
~~~~~~~~~~~~~~~~~~~~~~~~~~~~--> 
```

Controlling which reports appear in the Test Reports view: Database search and filter

You can perform a database search that populates the list of reports in the Test Reports view (that is, only reports that meet the search criteria are listed).

If test reports are stored in the built-in Spirent iTest database

1. In the Test Reports view toolbar, click **Database Search**  .
2. On the **Search Reports, Search** settings dialog, specify any or all of the search criteria:

Result	The search returns reports with the specified execution result: **Pass**, **Fail**, **Abort**, **Indeterminate**, or **All** (all of the result types).
Time period	Specify the time period during which the test case was executed.: today,last 2 days, last month, last 6 months, last year, or all.
Test Case	Enter the test case name results you want to view. In addition you may view the Full test case URL and/or only the latest test case result. • Select Full test case URL to view the test case location. • Select Show only latest to view the latest results report.
Reset to default **Cancel**	• Click **Reset to default** to discard your changes and set new search criteria. • Click **Cancel** to discard the changes and close the dialog.

3. Click **OK** to save and apply the search settings, and close the dialog. The view gets populated with the reports that meet the search criteria.

4 At this time, you have the option to apply a further filter to the results. See "Applying a filter to the reports listed on the Test Reports view" on page 441.

If test reports are stored in an external database

1 In the Test Reports view toolbar, click **Database Search**  .

2 On the **Database Search Settings** dialog box, specify any or all of the search criteria.

Host	The search returns reports from the specified iTest host.
Group	The search returns reports from the specified group
Subgroup	The search returns reports from the specified subgroup
Result	The search returns reports with the specified execution result: **Pass**, **Fail**, **Abort**, **Indeterminate**, or **All** (all of the result types).
Time period	Specify the time period during which the test case was executed.
Test case	Show reports only for the specified test case. Type only the filename and not the .fftc extension. The * wildcard character is supported.
Show full test case URIs	Check the box to display the full URI of the test case on the Summary section of the Test Report editor. Uncheck to display only the test case filename. Default: unchecked
Show only latest	Check the box to display only the most recent test report. Uncheck to display all reports for the test case (up to the maximum number specified in the **Maximum number of reports to fetch** preference setting). Default: unchecked

3 Click **OK**. The view is populated with the reports that meet the search criteria.

4 At this time, you have the option to apply a further filter to the results. See "Applying a filter to the reports listed on the Test Reports view" on page 441.

Applying a filter to the reports listed on the Test Reports view

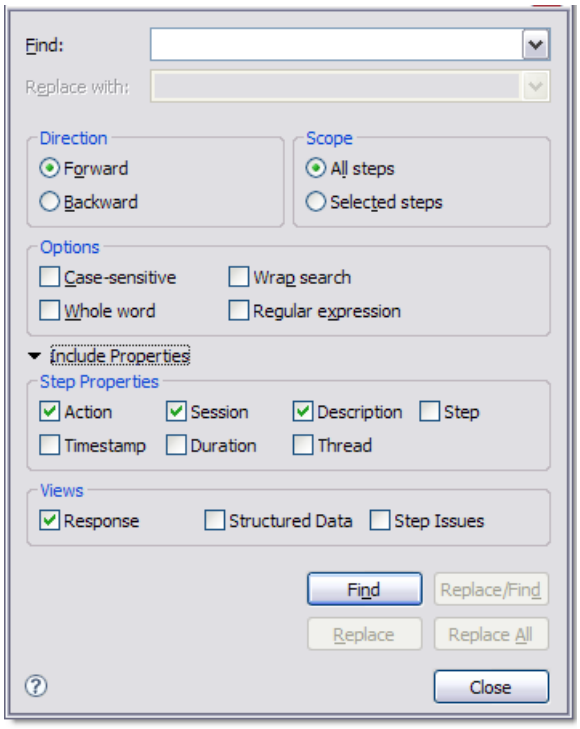
Once the database search returns the list of reports that meet the search criteria, you can further limit the reports that appear in the view by typing part or all of the associated test case name into the **Filter** text box at the top. Only reports for test cases that “pass the filter” now appear in the list. To reset so that no filter is applied, delete the text.

Searching for text in a test report

While viewing a test report, use the **Test Report Find** dialog box (**Ctrl-F** or **Edit > Find**) to find text in the report and in the views that provide information about test results: Response view, Step Issues view, and Structure view.

- You can specify which columns of the report to include or exclude from the **Find** process. In addition, you can include or exclude particular views.
- When a match is found in the report steps, the matching text is highlighted.

- When found in the Response view, the view is displayed and the matching text is highlighted.
- When found in the Structure view or Steps Issues view, the view is displayed and the row is highlighted.
- The status bar provides information about what was found.
- Because test reports are read-only, **Replace** operations are not supported.



Find	Specify the text to find. You can provide specialized text — see the **Options** properties.
Replace With	Because test reports are read-only, **Replace** operations are not supported.
Direction	Specify whether to search Forward or Backward from the cursor position. See the **Wrap search** option. Forward searches always begin at the beginning of the **Executed Steps** section of the report. **Note** You cannot search backward when a single step is selected.
Scope	**All steps**: Search all rows in the report. **Selected step**: Search all properties and views for the selected step and its child steps only.

Options	**Case-sensitive:** Matches must use the identical case as the text in the **Find** field. Does not apply if you check the **Regular expression** checkbox. **Whole word**: Matches must be the text in the **Find** field surrounded by whitespace. Does not apply if you check the **Regular expression** checkbox. **Regular expression**: Interpret the text in the **Find** field as a regex. **Wrap search:** This option ensures that all specified items are searched in the case that you start a search in the "middle". When the **Find** process reaches the end of the document (**Direction** property = **Forward**) or the beginning of the document (**Direction** property = **Backward**), the search continues.
Include Properties	Check an item to perform the **Find** process in the selected column of cells. Default: **Action**, **Session**, and **Description**. • Each match may include all or a portion of the text in the cell. • The **Find** process can find text in only the first line of multi-line commands (that is, text cannot be found in the body of a multi-line command). • Matches may not span property (cell) boundaries, with the following exception: The **Description** field is not actually a property but is sometimes a collection of property settings. (For example, for CLI session types, **Description** displays the content of the **Command** property.) For the **Find** process, **Description** is considered to be a single property, so all affected properties settings are searched.
Include Views	Check a view to perform the **Find** process in the view. Default: Response view

Buttons

Find	Find the next instance of the text specified in the **Find** field. During the Find process, the button changes to **Cancel** to allow you to cancel the current search.
Replace/Find **Replace** **Replace All**	Because test reports are read-only, **Replace** operations are not supported.
Close	Stop the **Find** operation and close the dialog box.

Finding issues and errors in a test report

Test reports display the system-defined and user-defined **execution issues** that are generated during execution. Each issue that is generated during execution has a **severity** (**Information**, **Warning**, **OK**, and **Error**) and an associated text **execution message**. Hold the cursor over the icon to read the execution message that is associated with the execution issue. Execution issues and messages also appear in the Execution view and the Step Issues view.

For very long test reports, it might take some time to locate issues. **Go To Next Issue** and **Go To Previous Issue** in the toolbar make it easier to quickly locate and view issues and the associated execution messages.

Comparing (“diffing”) two test reports

When you compare two test reports, Spirent iTest generates a ***comparison test report*** that identifies the differences and then opens the report in the **Test Report Comparison** editor.

To compare two reports

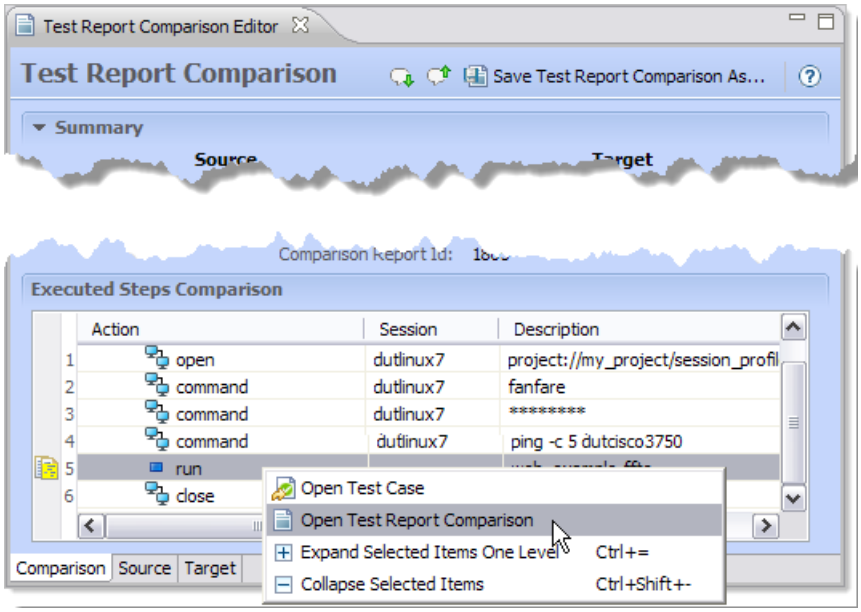
1. In the Test Reports view, select the reports that you want to compare (Ctrl+click).
2. Spirent iTest supports two modes for generating the report:
   - Compare two selected reports: Select the two test reports (use Ctrl+click). Right-click the selection and select **Compare Test Reports**.

Note Comparing test reports that were generated by different test cases may not be meaningful, so Spirent iTest displays a warning dialog box.

   - For a test case that failed, compare the failing run with the last (most recent) passing run — or the opposite: compare a passing run with the last failing run: Right-click the report and select **Compare with Latest Passing/Failing Test Report**
3. Spirent iTest generates the diff report and opens it on the **Comparison** page of the **Test Report Comparison** editor. The differences are marked by a diff icon in the **Executed Steps Comparison** section.

To compare reports for child test cases (tests executed by ‘run’ steps)

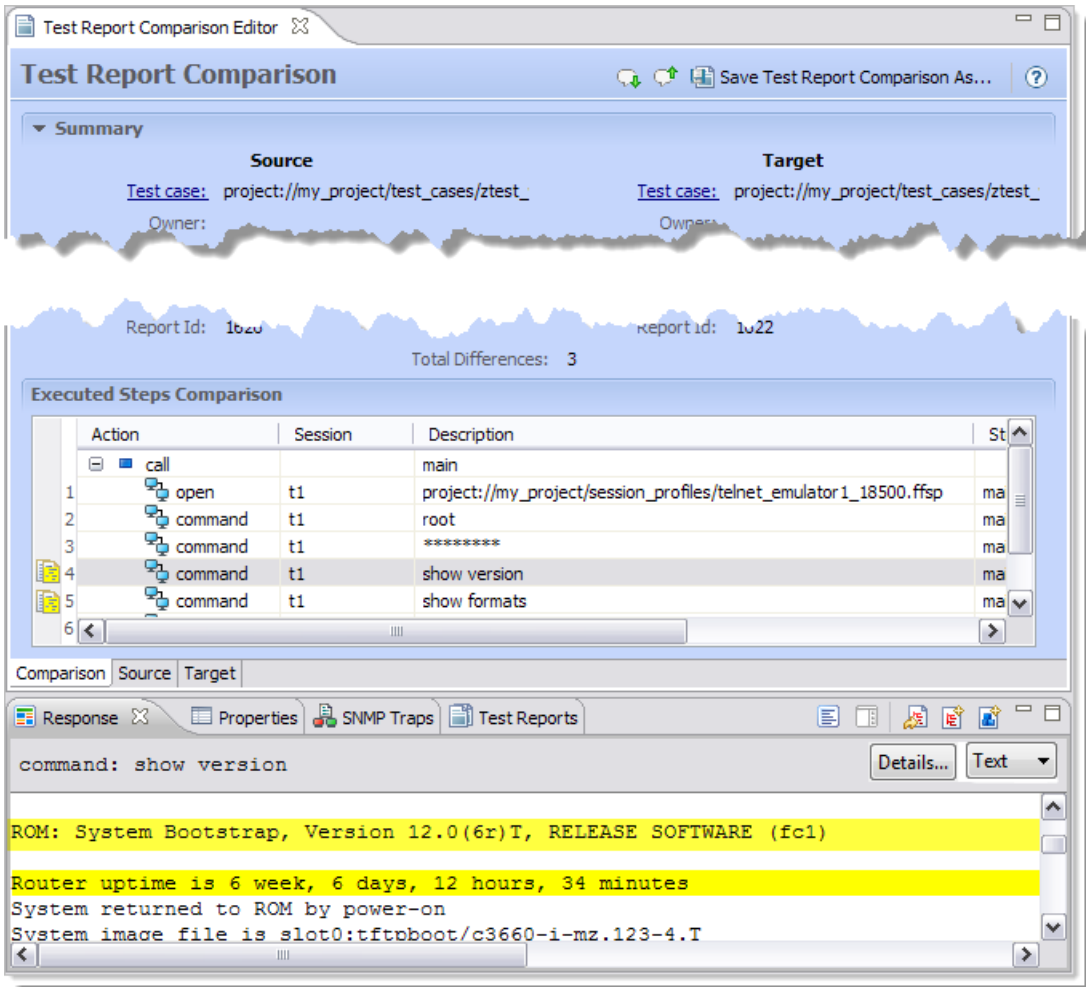
1. First generate the Test Report Comparison for the parent test cases, as described in "To compare two reports" on page 444.
2. In the **Test Report Comparison** editor, right-click the **run** step and select **Open Test Report Comparison**.



A new report opens in a new instance of the **Test Report Comparison** editor, comparing the two runs.

Working in the Test Report Comparison editor

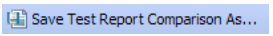
The editor has the following pages:



- **Comparison** page:
  - The **Summary** section displays identifying information about the two test reports. The older of the two reports is the “**Reference**” report and the newer report is the “**Target**”.
  - The **Executed Steps Comparison** section displays a diff icon for steps that differ between the reference report and the target report. Steps can differ in the following ways:
    - **Text**: If the step does not have a response map associated with it, then Spirent iTest compares the text of the response (for example, the target report includes a line that does not appear in the reference report).
    - **Query**: If the step has a response map associated with it, then Spirent iTest compares the queries. This type of comparison is very helpful in identifying subtle differences between software releases (for example, it can ignore unimportant diffs like timestamp and focus only on items that are identified with a query in the response map). For details, see "Examining queries for differences" on page 446.

        - **Execution flow**: If the two executions diverge (for example, an **if** step took the **True** path in the first execution and the **False** path in the second) then the comparison stops at the point of divergence.
    - **Response view**: When you select a diff step, the Response view displays the reference response on the left and the target response on the right, separated by the | character. The Response view highlights the lines in the response that differ between the reference report and the target report.
    - **Queries view**: When you select a diff step that has a response map associated with it, the Queries view identifies differences using the diff icon  . For details, see "Examining queries for differences" on page 446.
    - **Execution view**: The Execution view displays the list of differences marked by the diff icon  . Double-click a difference to select the corresponding difference on the **Comparison** page of the **Test Report Comparison** editor.
- **Reference** page: Displays the standard test report for the reference execution of the test case
- **Target** page: Displays the standard test report for the target execution of the test case

Toolbar

- For very long reports, it might take some time to locate differences. **Go To Next Diff**  and **Go To Previous Diff**  in the toolbar make it easier to quickly locate and view differences and the associated execution messages.
- You can save comparison test reports as HTML, PDF, Text, XML, or XML_RAW files by clicking **Save Test Report Comparison As** 

. See "Sharing comparison test reports as files" on page 446

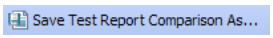
Examining queries for differences

If the step has a response map associated with it, then Spirent iTest compares the queries. This type of comparison is very helpful in identifying subtle differences between software releases (for example, it can ignore unimportant diffs like timestamp and focus only on items that are identified with a query in the response map).

Sharing comparison test reports as files

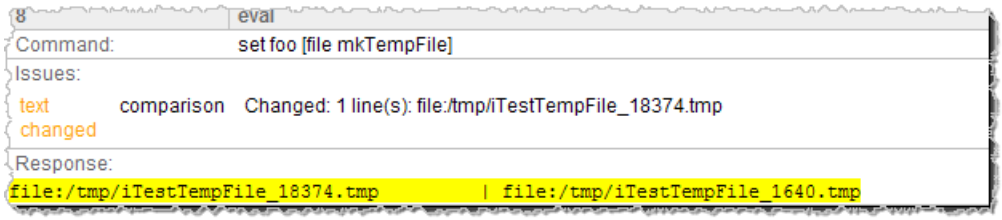
As with test reports, you can save comparison test reports as HTML, PDF, Text, XML, or XML_RAW files.

To save a comparison test report as a file

1. While working in the **Test Report Comparison** editor, click **Save Test Report Comparison As** 

in the toolbar.
2. Follow the instructions for saving test reports as files: "Sharing test reports as files" on page 449.

In this example **Test Report Comparison** report, we display the entry for a single step. The response to the command is different in version 4.8 than in version 4.7:



Sharing Test Reports

Configuring Spirent iTest to save test reports to an external database

By default, Spirent iTest saves test reports to a database within your workspace. Spirent iTest users that use other workspaces might find it cumbersome to access test reports in your workspace. To make it easier for everyone on your team to share report data, you can configure Spirent iTest to save test reports to an external centralized database that is accessible to all Spirent iTest and iTestRT users rather than to the default 'standalone' database within your workspace.

Note To use an eternal database, your system administrator must first set up a database server. For instructions, see the chapter on "Setting up an external database for storing Spirent iTest Test Reports" in the *Spirent iTest Installation Guide*. In addition, when upgrading Spirent iTest, the administrator must also update the database schema (also described in the *Spirent iTest Installation Guide*).

To configure Spirent iTest to save reports to the external database

Follow these instructions after your system administrator has configured a database server for use with Spirent iTest.

1. Install Spirent iTest on a client computer.
2. Start Spirent iTest and click **Window > Preferences**.
3. On the **Preferences** page, in the **iTest** group, navigate to **General > Test Reports > Test Report Database**.
4. Select **Use an external database to store test reports** and then specify the following settings (the information should be available from your system administrator).

   Note By default iTest reports are saved to the embedded database.

   You can configure the connection in one of the following ways:

- For any database type other than **'Other'**:

  Specify the **Database type**, **hostname or IP address** and **IP port** of the database server, the **database/catalog name or SID** and, if required, the **User ID / Password** credentials.

  or

- For a database type of **Other:**

  Specify the **Java class for the custom JDBC driver**, **JDBC connection string**, and, if required, the **User ID / Password** credentials.

In any case, you must specify the **User ID / Password** credentials for the account

Database type	Specify the type of database server Default: MySQL
Database server address / Hostname	Specify the hostname or IP address of the database server. Default: localhost
Database server port number	Specify the IP port of the database server. If you specify a value for **Database type**, then you can leave this property blank to use the default port for the specified database type. Default: 5340
Database/Catalog name/SID	Optional. Specify the database/catalog name or SID. To connect to Oracle Database Express Edition, set the SID as **xe**. Default: reports
User ID	Optional. Specify the username used to connect to the database server.
Password	Optional. Specify the password for the **User ID** account.
JDBC connection string	Optional. Enter the URL of the JDBC connection. For example, **jdbc:mysql://[host][:port]/[database]** See the topic on "Adding a custom third-party JDBC driver to iTest" in the *iTest Installation Guide*. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
Driver class	Optional. Specify the Java class for the custom JDBC driver. For example, **com.mysql.jdbc.Driver** See the topic on "Adding a custom third-party JDBC driver to iTest" in the *Spirent iTest Installation Guide*.

Important Click **Test Connection** to confirm the settings.

5 To save and apply the settings, click **OK** and then exit and restart Spirent iTest.

Sharing test reports as files

To ensure high performance, Spirent iTest does not, by default, automatically save test reports as files. So, to share a test report, you must first export it as a file. There are two options:

- **Export or auto-export the report as an HTML, PDF, Text, XML,** or **XML_RAW**. This is useful for sharing with someone who does not use Spirent iTest.
  - See "Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file" on page 451.
  - You can set a preference to auto-export a test report whenever a test case executes. See "Setting preferences for Test Reports" on page 455 for details on configuring auto-save.
- **Export the report as a compressed Spirent iTest Test Report file**. When you export a report as a compressed file, other Spirent iTest users can open it in the Test Report editor. You can email the resulting file to a coworker or tell the coworker the path of the file (typically, in a shared workspace under revision control). The coworker then uses Spirent iTest to import the file into their workspace. See "Exporting a test report as a compressed file (fftz format)" on page 451.

  The person importing the compressed file will follow the instructions in "Importing Spirent iTest test reports that are in compressed file format (fftz)" on page 453.

Tip You have the option to save only selected sections of any report when you export it. See "Customizing the content of a test report" on page 459.

For descriptions of the various formats, see "Test report file formats" on page 449.

Test report file formats

Important In test reports that have been saved in HTML, PDF, Text, XML, or XML_RAW formats, **OK** severity issues are listed as “pass” and **Error** as “fail”.

On the test report’s **Response** section, iTest displays response format as **JSON** or **Text** form (as auto-detected).

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.

  Only text indentation is applied (to HTML, HTML/JSON, PDF, and customized reports) and not the text color.
- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, “JSON Editor”).

You can specify the following format for test report files:

HTML, PDF, Unicode text format reports

Reports include information on the parameters used in the test case in a table below the summary information at the top of the report. Parameter information includes name, description, value, and the source of the value used.

HTML only: If the report includes snapshots, then Spirent iTest creates an **images** subfolder for the snapshot files. See "Screen snapshots in HTML-format reports" on page 429 for an example report that includes an **open** step that returns a snapshot of a web page and a **describe** step that returns a snapshot of a button on the page. You must include the **images** subfolder if you share the report.

See "Example HTML test report as viewed on a browser" on page 428.

XML and XML_RAW format reports

The report is a single XML document containing all of the fragments in the context of an overall valid XML document.

The XML test report also includes a copy of the entire test case document object under the **testCaseDetails** tag, but excludes the **Procedures** element. As a result, all general information about the test case is available, including parameters. If used, the entire topology or testbed, is placed under the **testbedDetails** tag.

One way to add runtime information into a test report is to create a parameter in the test case to hold the information. Add an **eval** step to the test case with a **command** of **[gset parameters/param_name value]** to update the value of that parameter at runtime.

The predefined XSLT templates are stored in **reports<version>** folder in the **resources** project.

To use report formats that you used with previous versions of Spirent iTest, copy the folders under the **reports** folder into the **reports<version>** folder.

> **Advanced users:** If you use a custom format for HTML files and have placed XSLT templates in the **reports<version>** folder under the **resources** project, then the format appears as an option in this list box.

Compressed file format

When you export a report as a compressed file, you can share it with other Spirent iTest users for opening in the Test Report editor. See "Exporting a test report as a compressed file (fftz format)" on page 451.

Auto-saving every test report to an HTML, PDF, Text, XML, and/or XML_RAW file

You can configure a preference setting to cause Spirent iTest to auto-export each test report in any or all of HTML, PDF, Text, XML, and/or XML_RAW formats. See "Setting preferences for Test Reports" on page 455 for details on configuring auto-save.

Format descriptions appear in "Test report file formats" on page 449.

Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file

Important In test reports that have been saved in HTML, PDF, Text, XML, or XML_RAW formats, **OK** severity issues are listed as “pass” and **Error** as “fail”.

You can save a test report from either of the following starting points:

- While viewing a test report in the Test Report editor, click **Save Test Report As** 

.
- In the Test Reports view, select one or multiple test reports. Right-click and select **Save Test Report As** (or click ).

Format	Specify the format of the test report file, **HTML, PDF, Text, XML,** or **XML_RAW**. Format descriptions appear in "Test report file formats" on page 449.
Destination Folder	Specify where to save the report. **Workspace folder:** Save the file in a folder in the current workspace. iTest users that use the workspace will see the folder (and the new test report file) in the iTest Explorer. The default folder is **saved_test_reports**, but you can specify any folder in the workspace or specify a new folder. If you specify a new folder, iTest creates it and then adds the report file to it. **File system folder:** Save the file in the specified folder (typically, outside of the current workspace). If the file is saved outside of the workspace, iTest users that use the current workspace will not see the folder (or the new test report file) in the iTest Explorer. Instead, use the operating system's methods for accessing files.
File name	iTest provides a unique, time-stamped name for the report (**test_case_name_time_stamp.file_extension**). You can modify the name as needed. If you selected multiple test reports in the Test Reports view, iTest will use the default names for the resulting files.
View the report in an editor after saving	Check the box to display the report in a text editor after you save it. Text and XML files open in a text editor. HTML files open in the default iTest browser.

Exporting a test report as a compressed file (fftz format)

Note This section describes how to share reports with Spirent iTest users that use different workspaces from yours. This operation is not necessary if:

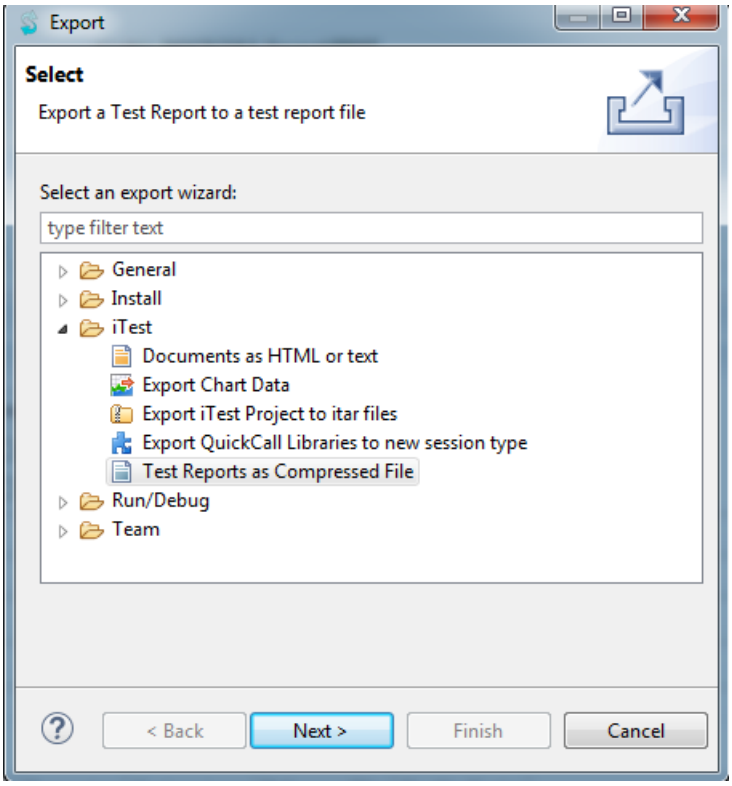
- You have configured Spirent iTest to save test reports to an external database rather than to the default database within your workspace. See "Configuring Spirent iTest to save test reports to an external database" on page 447.
- You share a report in another format with persons who do not have access to Spirent iTest. See "Manually saving a test report as an HTML, PDF, Text, XML, and/or XML_RAW file" on page 451.

When you export a report as a compressed file, you can share it with Spirent iTest users (that use different workspaces) for opening in the Test Report editor.

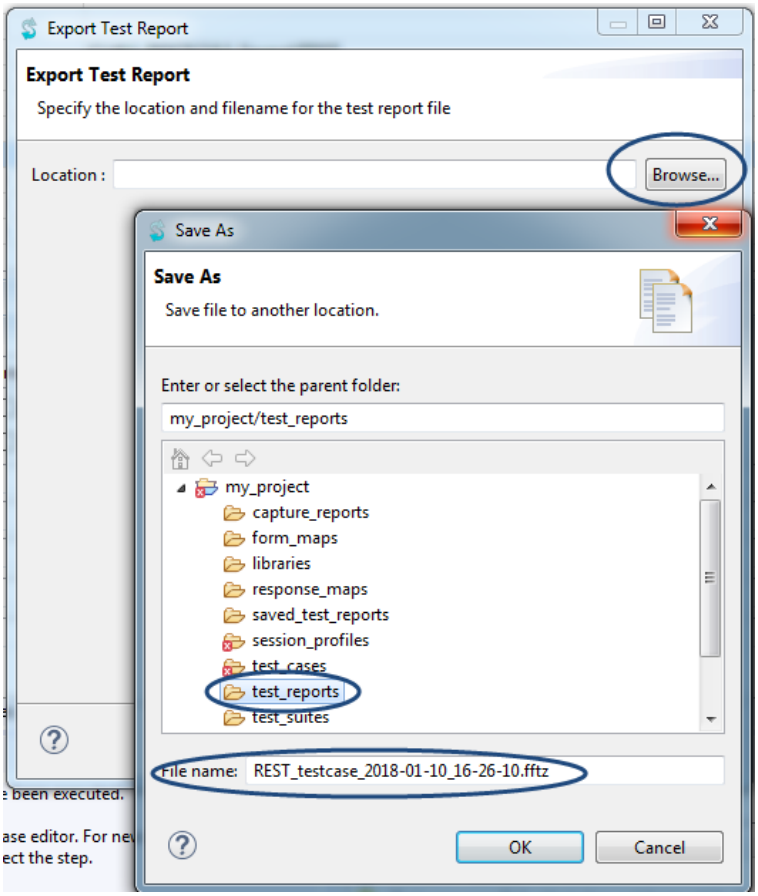
1 Select the report to export and start the Export wizard. Use any of the following methods:

- In the Test Reports view, select the report. On the main menu, click **File > Export**.
- While working on a test report in the Test Report editor, click **File > Export** on the main menu.
- For a report saved in any of the other formats (PDF, Text, XML, or XML_RAW): In the Spirent iTest Explorer, right-click the report and select **Export**.

2 On the Export wizard's **Select** page, select **iTest > Test Reports as Compressed File**. Click **Next**.



3 Specify the location in your workspace to export the file to. Click **OK**.

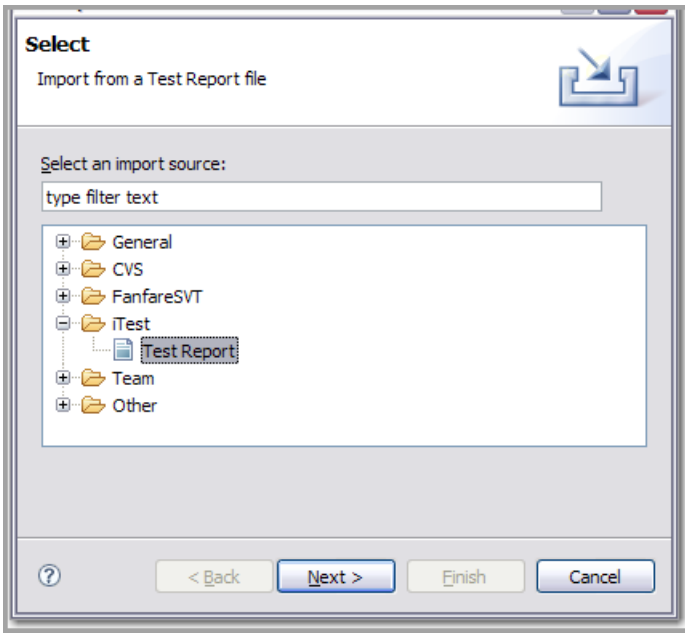


4 On the next page, click **Finish**.

Importing Spirent iTest test reports that are in compressed file format (fftz)

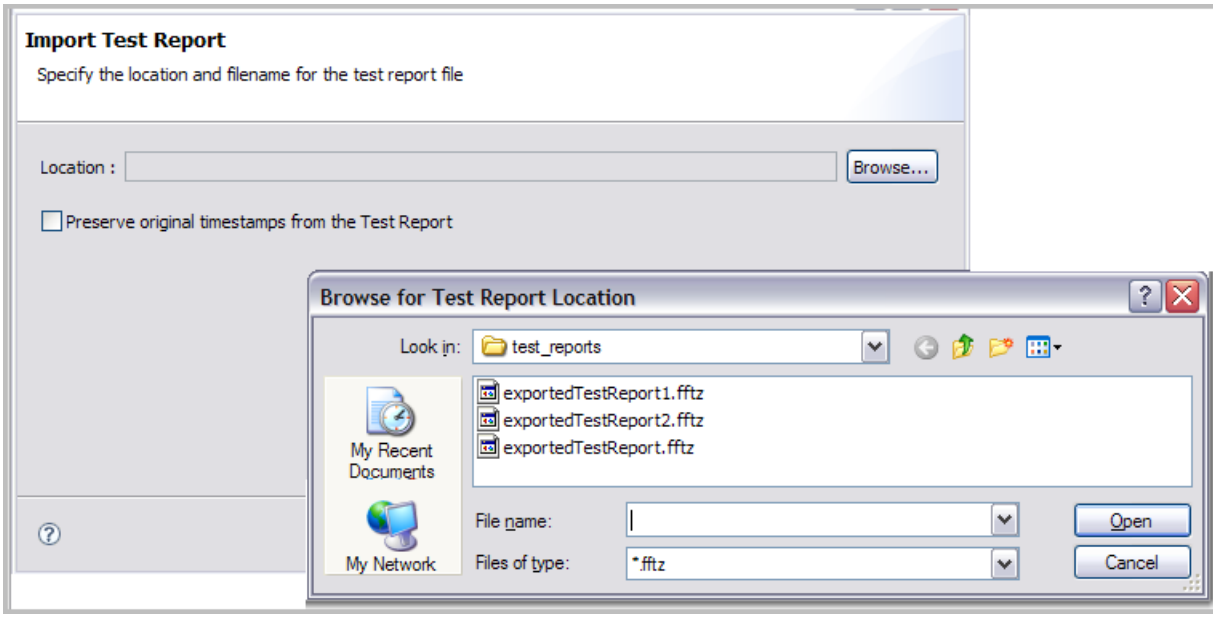
Follow these instructions to import an Spirent iTest test report that had previously been saved as a compressed file in **fftz** format (the compressed format for reports as generated using the Export wizard). See "Exporting a test report as a compressed file (fftz format)" on page 451 for instructions on exporting a report to a compressed file.

1 Click **File > Import** and select **iTest > Test Report**. Click **Next**.



2 On the **Import Test Report** page:

a For the **Location** box, browse to the directory that holds the file to be imported. Select the file and click **Open**.



b Optional: Check **Preserve original timestamps** if the timestamp data is important for your purpose.

c Click **Finish**.

3 The file is imported and added to the end of **Today's** reports in the Test Reports view. Double-click to view the report in the Test Report editor. For more details about working in the Test Reports view, see "Test Reports view" on page 435.

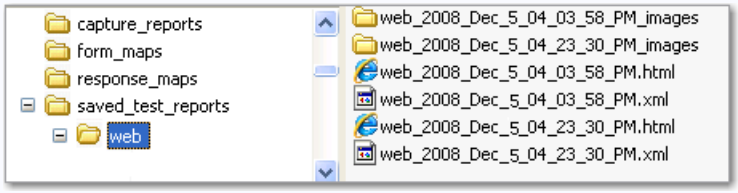
Setting preferences for Test Reports

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > General > Test Reports.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > General > Test Reports > Auto-Export Test Reports

Export test reports after execution	Check the box to cause iTest to generate a test report document after each test execution. • One report is generated for each execution. • Report files are named with the test case name and a timestamp. • You specify the format of the report documents (HTML, PDF, Text, XML, and/or XML_RAW) using the **Formats to generate** property. • You specify the location to save reports using the **Folder for test reports** property If you uncheck the box, then iTest does not automatically generate report documents. In either case, you can manually save any test report as a file as described in "Uses for test reports" on page 426.
Export test reports for test cases executed in 'run' steps	Check the box to include test report data for any test case that is executed as the result of a **run** step in the parent test case.

Formats to generate	If you checked **Export test reports after execution**, then specify each document format to generate. You can specify any or all formats (**HTML, PDF, Text, XML, XML_RAW)**. For format information, see "Test report file formats" on page 449.
Folder for test reports	Specify the folder that should receive all auto-generated report documents. The default folder for test report documents is: **project:///saved_test_reports/<test_case_name>/<test_case_name>_<timestamt>** Test reports for session types that use browsers (Swing and Web) can include thumbnail images of screen snapshots. When the user clicks a thumbnail, the full-size image appears. The full-size image files for reports are saved into the **<report subdirectory>/<test_case_name>/<test_case_name>_<timestamt>_images** subdirectory. In this example, we executed the test case named **web** on two different dates, we specified HTML and XML format reports, the test case includes steps that generated snapshot images, and we used the default subfolder names: 

Spirent > General > Test Reports > Database Aging

To ensure that the database of test reports does not become too large, you can specify that old reports should be deleted when they reach a particular “age”.

Delete reports older than	**Note** This setting applies only to the default iTest “standalone” database within your workspace. When a test report exceeds the specified age in days, it is deleted. **Note** A report will be deleted only after the current execution of the test case is completed and the next execution of the same test case starts. For a recursive test case (a test case that includes a **run** step that executes itself), the test reports are never deleted. Exit and restart iTest to apply a new setting. Default: 30

Editors > Test Reports > Database

On the **Test Report Database** preferences page you may enable or disable response compression in a test report database (internal or external) and configure Spirent iTest to use an external database to store the test reports.

Response compression

Usually test case responses occupy a lot of space in a test report database, internal or external. To make a test report database more compact one can enable response compression in a test report database. By default response compression is enabled.

Enable response compression	Checkbox to cause iTest to compress responses when storing them in a test report database

Using an external database to store test reports

To make it easier for everyone on your team to share test report data, you can configure Spirent iTest to save test reports to an external centralized database that is accessible to all Spirent iTest and iTestRT users rather than to the default “standalone” database within your workspace. See "Configuring Spirent iTest to save test reports to an external database" on page 447.

Spirent > General > Test Reports > Test Report Tags

These properties determine the database tags that are associated with test reports.

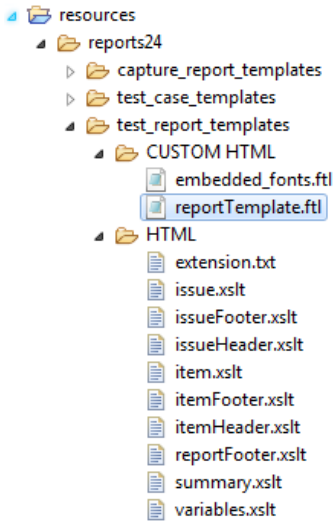
Host	Optional. Default: iTest tags each test report with the hostname where the test case executes. If you specify text here, then each report is tagged with the text instead of the actual hostname. • You can use the value to search for test reports in the database. • The text appears in the **Host** column on the Review Test Reports activity page and on the Test Reports view.
Group	Optional. External database only. If you specify text here, then: • The value acts as a parent to the optional **Subgroup** value. • Each report is tagged with the text. • You can use the value to search for test reports in the database. • The text appears in the **Group** column on the Review Test Reports activity page and on the Test Reports view. Default: [blank]
Subgroup	Optional. External database only. If you specify text here, then: • The value acts as a child of the **Group** value. • Each report is tagged with the text. • You can use the value to search for test reports in the database. • The text appears in the **Subgroup** column on the Review Test Reports activity page and on the Test Reports view. Default: [blank]

Spirent > General > Test Reports > Test Reports View

Maximum number of reports to fetch	Specify the maximum number of test reports to display on the Review Reports activity page and the Test Reports view. Default: 1000

Customizing the Standard HTML test report template

You can customize the iTest HTML+JSON reports, personalize the content, include your branding, and select what should be included included in the report. The resulting test report template can be chosen when exporting the testreport in iTest. You may also distribute the template to your co-workers to ensure that everyone uses the standard report template (You may also use the customized template to generate reports in Velocity)..



To create/modify a custom template follow these steps.

1. In iTest Explorer, browse to the **/resources/reports24/test_report_templates/**
2. Copy the folder with the template to be modified (e.g., HTML+JSON folder) to the same directory: **/test_report_templates/**
3. Rename the copy of the template folder to create a new custom template (e.g., **CUSTOM HTML**).

   Note Report name is inherited from folder name.

4. Modify the **.ftl** inside the template folder.

   To use a custom logo, add your custom image to the **CUSTOM HTML** folder and rename the image you added as **customImage.png**.

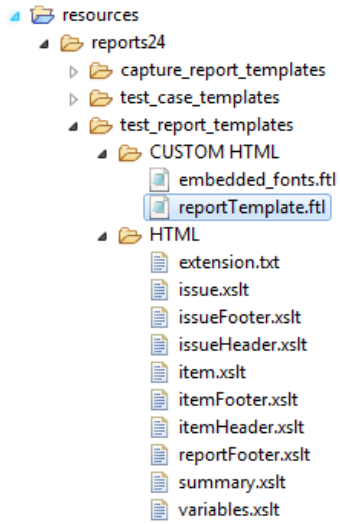
   Note The name **customImage.png** is the default/fixed name and iTest uses this image name as the custom logo to generate/print the custom report.

5. Modify report content as in "Customizing the content of a test report" on page 459.

6 Now, whenever you save a test report by clicking **Save Test Report As**, **CUSTOM HTML** appears as an option in the **Format** drop-down list. Select **CUSTOM HTML**, and, the resulting report displays as per your template.

Customizing the content of a test report

You can omit any section from an exported test report document by deleting its associated stylesheet from the **resources** project. The **test_report_templates** folder has a subfolder representing each file type that you can save:



For example, to save only the **Summary** portion of HTML reports:

1 First, make a copy of the **HTML** folder (to a folder named **my_custom**, for example).

2 In your folder, delete all of the **xslt** files in the **HTML** folder, except **summary.xslt**.

3 Now, whenever you save a test report by clicking **Save Test Report As**, **my_custom** appears as an option in the **Format** drop-down list. Select **my_custom**, and, in the resulting report, only the **Summary** section of the report will be included.

Creating a Customized HTML Report

You may download the default template provided by iTest, customize it as required using the provided template data model **Spirent-APT-Report-DataModel.pdf** (on Spirent Knowledgebase) and upload the custom template. You may also upload custom images as the logo to be used with your custom template.

To create/modify a custom template follow these steps.

1 In iTest Explorer, browse to the **/resources/reports24/test_report_templates/**

2 Copy the folder with the template to be modified (e.g., HTML folder) to the same directory: **/test_report_templates/**

3 Rename the copy of the template folder to create a new custom template (e.g., **CUSTOM HTML**).

Note Report name is inherited from folder name.

4 Modify the **.ftl** inside the template folder as per the instructions in **Spirent-APT-Report-DataModel.pdf** (on Spirent Knowledgebase.)

To use a custom logo, add your custom image to the **CUSTOM HTML** folder and rename the image you added as **customImage.png**.

Note The name **customImage.png** is the default/fixed name and iTest uses this image name as the custom logo to generate/print the custom report.

5 Modify report content as in **Spirent-APT-Report-DataModel.pdf** (on Spirent Knowledgebase).

Now, whenever you save a test report by clicking **Save Test Report As**, **CUSTOM HTML** appears as an option in the **Format** drop-down list. Select **CUSTOM HTML**, and, the resulting report displays as per your template.

For advanced XSLT options, see "Using Spirent iTest's XSLT stylesheets to format reports" on page 461.

Using Spirent iTest's XSLT stylesheets to format reports

Spirent iTest uses the Java XML transformer and XSLT stylesheets to transform Spirent iTest XML documents to various document types. XML_RAW, Text, PDF, and HTML styles are supported for test reports, capture reports, and test cases.

Additional XML styles are provided for test reports.

You can export test reports in a format that matches a published schema. This ensures that any tools that you create to extract the test report data will work with future versions of Spirent iTest.

- The "XML" choice produces a Spirent iTest XML test report file that adheres to the schema in **resources/reports<highestNumber>/test_report_templates/XML/test_report.xsd**
- The "XML Raw" choice is the Spirent iTest 3.1 full report format.

The stylesheets are located in the workspace under the **resources** project. The project must be open for the transformations to work. They are created the first time you try to export a document.

Spirent iTest 5.0 (and above) directory structure

In Spirent iTest 5.0 (and above versions), the XSD is located in **resources/reports24**. The capture report and test case templates are the same as for the previous versions. The directory structure for the XML format is as follows.

resources/reports24/test_report_templates/XML/

extension.txt
issue.xslt
issueFooter.xslt
issueHeader.xslt
item.xslt
itemFooter.xslt
itemHeader.xslt
reportFooter.xslt
summary.xslt
test_report.xsd

Note The directory for the PDF format includes files: **report.fo** and **variables.xslt**.
The directory for the HTML format also includes file: **variables.xslt**.

Spirent iTest 4.0 directory structure

In Spirent iTest 4.0, the XSD (**test_report.xsd)** is located in:

resources1/reports18/test_report_templates/XML

The directory structure for test report templates is the same as for Spirent iTest 3.2.

Spirent iTest 3.2 directory structure

In Spirent iTest 3.2, the XSD is located in **resources/reports5**. The capture report and test case templates are the same as for Spirent iTest 3.1. The directory structure for the 3.2 XML format is:

resources/reports5/test_report_templates/XML/

extension.txt
issue.xslt
issueFooter.xslt
issueHeader.xslt
item.xslt
itemFooter.xslt
itemHeader.xslt
reportFooter.xslt
summary.xslt
test_report.xsd

Note If you have made changes to the Spirent iTest stylesheets, Spirent iTest will not overwrite the changes. However, if you want to use the changes, you must copy them to the appropriate folder under **reportsxx** (where **xx** varies depending on the Spirent iTest version).

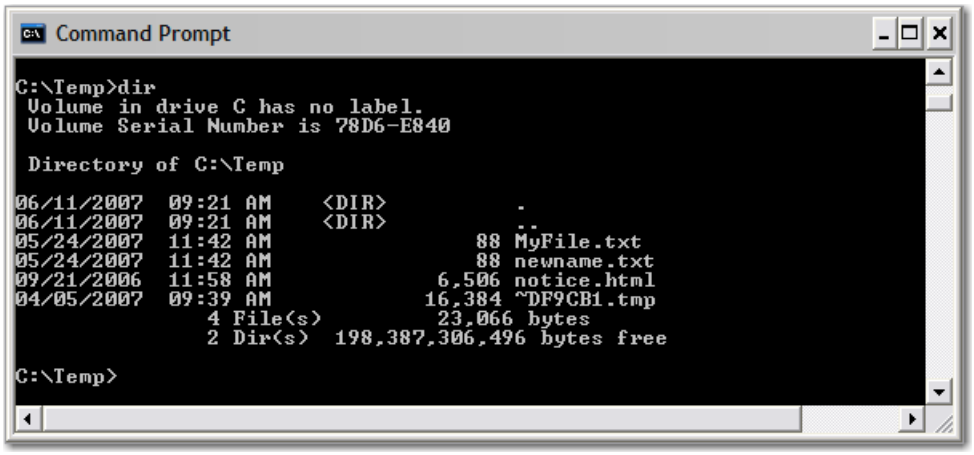
CHAPTER 22

Prompts (in CLI sessions)

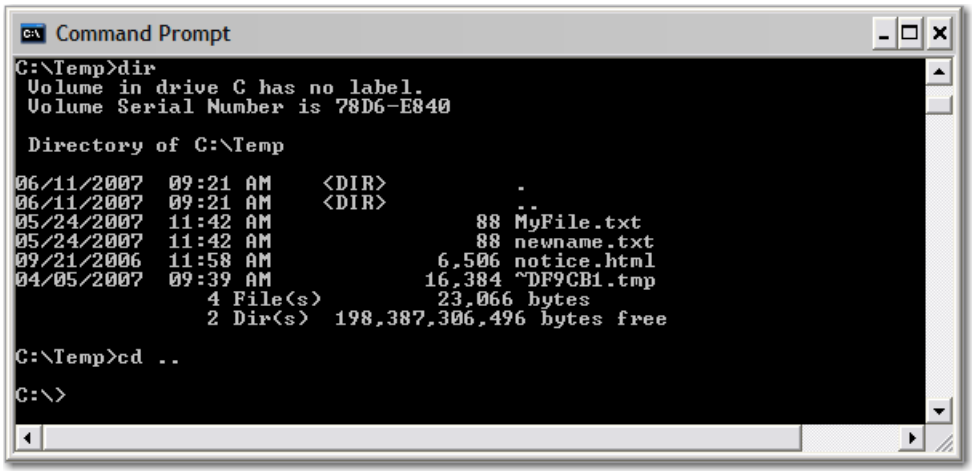
Overview: Prompts in iTest

When you interact with a device's command line interface (CLI) by submitting a command, you know that the response is complete and the device is ready to accept a new command when a prompt appears.

For example, on a Window PC's command line, we submit the `dir` command at the device's `C:\Temp>` prompt. The device responds with the directory contents as we commanded. The device then displays the `C:\Temp>` prompt to tell us that command execution is complete and that the device is now ready to accept a new command.



Now, let's move up a directory level by submitting a `cd ..` command. The device moves to the directory and then returns a prompt, but it's a different prompt entirely; the device has returned the `C:>` prompt.



How iTest distinguishes prompts from responses during execution

We humans have learned that both `C:\Temp>` and `C:>` are prompts, but how can iTest know that during execution? It turns out that iTest learns about prompts in the same way as we do:

Learning prompts during interactive (manual) testing

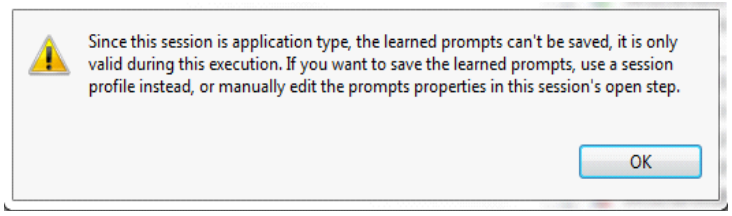
iTest identifies possible prompts by noticing when the session returns text and then goes silent for a significant period of time.

When you close a session, iTest starts the **Update Session Profile** wizard to show you the list of possible prompts that it noticed during the session. This gives you the opportunity to identify the text strings that actually are prompts. When you finish the wizard, the new prompt definitions are added to the session profile. If needed, you can then use the Session Profile editor to customize the property settings for the prompt.

Learningexecution

During test case execution, if the session channel becomes idle and the text on the line is not a defined prompt, then the status bar presents a **Learn this prompt** link that opens a dialog box to enable you to tell iTest that the text is a prompt. If you do not click the link, the step eventually times out and the test case continues execution.

Note When executing a session that is application type (for example: application:com.fnfr.svt.applications.ssh), a warning dialog displays as shown below, since these learned prompts are currently not saved.



How iTest stores prompt definitions

Each session profile or testbed device includes a list of prompt text strings and **Completion** property settings that tell iTest how to determine that the device is finished sending a response and the text on the command line is indeed a prompt.

iTest includes built-in definitions for some of the most common prompt strings (login prompts, password prompts, typical command line prompts like > and **%**, and the “more” prompt), so you typically will not have to customize prompt definitions.

Accessing prompt text for analysis

iTest stores prompt text separately from response text. You cannot combine the prompt text with the response text (for example, to analyze the combined text). Prompt text appears in the following locations:

- Test reports
- Prompt text is stored in structured data of the response. The Structure view displays the text in its appropriate place in the structured response.

Teaching iTest the prompts to expect during execution

iTest distinguishes response text from prompt text by checking whether text on the command line matches with anything in its list of known prompts (just like you do during manual testing!).

Let’s first discuss how iTest responds to a known prompt (either a built-in prompt or a prompt that you have defined). During execution, if the following conditions are met, then iTest treats the text as a prompt:

- The criteria specified by the **Completion** properties are met. The most important default setting is “the session channel is idle longer than the time specified by the **Idle channel interval** property for the session” (default value 100 msec).
- The last line of the response matches one of the prompt text strings specified for the session profile or device (the string can be wildcard text or a regular expression)

In contrast, either of the following conditions might cause an executing step to fail:

a No text appears at the command line for an extended period

This situation is avoided by telling iTest how to determine when a step is complete. You do this by configuring the **Completion** properties appropriately, as discussed in “Preparing for missing or unknown prompts during automated execution: Completion properties” on page 467.

or

b Unexpected text followed by a delay (silence on the session channel)

This situation is avoided by ‘teaching’ iTest that the text represents a prompt (when appropriate).

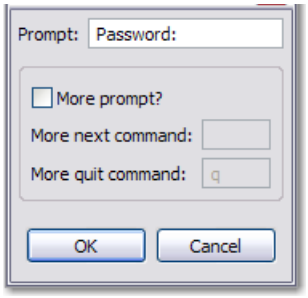
Teaching prompts during manual testing

During interactive testing, iTest might identify text that “acts like” a prompt (that is, it is the last text in a response and there is no additional response for a significant period of time). If

iTest notices possible prompts, it starts the **Update Session Profile / Testbed Device** wizard when the session ends. See "Using the 'Update Session Profile / Testbed Device' wizard" on page 79.

Teachingexecution

During test case execution, if the channel becomes idle and the text on the line is not a defined prompt, then the status bar presents a **Learn this prompt** link. When you click the link, the **Learn Prompt** dialog box enables you to tell iTest that the text is a prompt.



- If you click **OK** to identify the text as a prompt, then the test case result is unchanged and execution continues immediately with the next step.

  If the text represent a "more" page continuation prompt, you can set the **More** settings in a similar way as in the Session Profile editor. default values: **next page** character is a space and **quit** is the letter **q**.

  When execution finishes, iTest opens the **Update Session Profile / Testbed Device** wizard so you can add the prompt definition to the session profile or tested device. See "Using the 'Update Session Profile / Testbed Device' wizard" on page 79.

- If you click **Cancel**, then the step continues to wait for the prompt until it eventually times out. At that time, the step fails and execution continues.

Telling iTest about special situations like numbered prompts

iTest provides the **Update Session Profile** wizard to make it easy to add prompts that were discovered during execution, but how do you let iTest know that the prompt can change after each command (numbered prompts) or that the text of the last command appears after the prompt? How do you modify a prompt definition so that iTest does not react incorrectly to a response that requires a long time to appear at the command line?

You use the set of **Prompt** properties for the session or device to customize a prompt definition, as described in "Editing prompt definitions" on page 467 for instructions.

Preparing for missing or unknown prompts during automated execution: Completion properties

You use **Completion** property settings to define when the execution of a step should be considered complete. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- A step may include an analysis rule that examines the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

The determination of when a step is complete is protocol-specific and is controlled by the settings of the **Completion** properties (described in a moment).

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

For CLI protocols, you can specify any of several conditions to define when the step is complete, for example the existence of certain text in the response or the time elapsed after sending the command. The default **Completion criteria** property setting is that the step is complete when:

a The session channel is idle for the time specified by the **Idle channel interval** property

and

b The last line of the response matches one of the prompt definitions specified for the session profile or device.

To configure session profile properties that specify how to interact with the user when delays occur, see the **Completion** property settings:

- **Command Prompt sessions**: "Terminal > Replay > Step Defaults > Completion" on page 884
- **Serial sessions**: "Terminal > Replay > Step Defaults > Completion" on page 953
- **SSH sessions**: "Terminal > Replay > Step Defaults > Completion" on page 1131
- **Telnet sessions**: "Terminal > Replay > Step Defaults > Completion" on page 1209
- **TL1 sessions**: "Configuring sessions and test case steps for TL1 devices" on page 1215

Editing prompt definitions

In this topic, we discuss how to allow for variations in prompt definitions to enable smooth automated execution.

Example: Unexpected character in the prompt

Some devices return prompts with the following formats (notice that an asterisk * character appears as part of the prompt string):

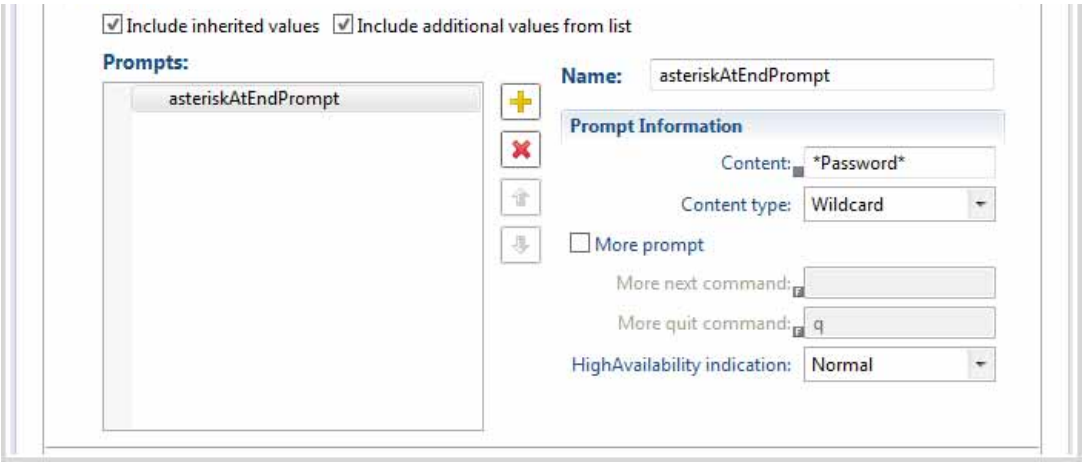
```
Enter Password:*
Re-enter Password to Validate:*
```

The **password** prompt definition that you might have created does not expect to see the * character as the last character on the line. So, to allow iTest to accept this prompt, we'll define a new prompt.

1 On the **Session Profile** editor **Start** page, perform these steps depending on the sessions type.

- For **SSH** and **Telnet** session types, open the **Prompts** property settings.
- For the **Command Prompt** and **Serial** sessions, click to open the **Session Properties** section, then open the **Terminal > Prompts** group of property settings.

2 Enter the required information as described in the steps below.



3 Check **Include additional values from list** to allow you to add a prompt definition. (For a discussion on inheriting prompt definitions from reference session profiles — the **Include inherited values** checkbox, see "About inheriting prompt definitions" on page 470.)

4 Now, click  to add a new prompt definition and name it **asteriskAtEndPrompt**.

We want iTest to accept prompts of the form ***Password:***, so we type that text as the value of the **Content** property.

The **Content Type** is **Wildcard** because we want iTest to interpret the * character to mean "any number of any character, including whitespace".

- All prompt definitions are case-insensitive.
- Leading and trailing whitespace is trimmed from any prompt text before iTest attempts to determine whether response text is a prompt.
- If you use regular expressions in the **Content** value, then set the **Type** property to **Regex**.
- If the prompt includes a space character or any whitespace in the body of the text, be sure to set the **Content Type** property to **Wildcard**.

  This new prompt definition enables iTest to now accept prompt text that matches any text (represented by the first wildcard *), followed by the text string "`Password:`", followed by any additional text (represented by the second wildcard *).
- Select **More prompt** property settings to enter more next command and more quit command. For these more prompts you may also select option to indicate High Availability: Normal, Master, Slave or Other.

Tip To avoid unnecessary interruption while testing, after you have added all prompt definitions to a session profile or device definition, you can uncheck **Learn prompts** in the **Capture** properties group.

Note The property settings discussed in this example are fully described for each CLI session type in the associated "Terminal > Prompts" section:

Command Prompt sessions: "Terminal > Prompts" on page 879

Serial sessions: "Terminal > Prompts" on page 947

SSH sessions: "Prompt" on page 1122

Telnet sessions: "Prompt" on page 1201

Example: Linux

When you start a new CLI session, you typically need to log in to the device. On a Linux device, login might look like this:

```
Red Hat Enterprise Linux WS release 4 (Nahant Update 2)
Kernel 2.6.9-22.EL on an i686
login:
```

The text `login:` tells you that a session has been established with the device and you can now provide the login information. We call this built-in prompt the **login** prompt. Similarly, after you enter the login information and press Enter, you receive the next prompt for providing the password (the built-in **password** prompt).

```
Red Hat Enterprise Linux WS release 4 (Nahant Update 2)
Kernel 2.6.9-22.EL on an i686
login: user01
Password:
```

Note, however, that the **login** and **password** prompts can vary for different devices.

After entering the password and pressing Return, another prompt appears:

```
[user01@fflinux1 ~]$
```

Let's call this the **normal0** prompt (it's the prompt that a user is most likely to see).

```
Red Hat Enterprise Linux WS release 4 (Nahant Update 2)
Kernel 2.6.9-22.EL on an i686
login: user01
Password:
Last login: Wed Jun 20 07:39:05 from officevpn
[user01@fflinux1 ~]$
```

In this example, the device's **normal0** prompt contains information that is specific to the particular device. It contains the username, the hostname, and the current working directory. If we change to a different directory using the `cd` command, you can see that the prompt changes:

```
[user01@fflinux1 ~]$ cd testplans
[user01@fflinux1 testplans]$
```

Example: Cisco

Let's look at the prompts on a Cisco device. Upon starting a session, no username is required, just a password. Therefore, this device does not require a **login0** prompt. The **password0** prompt is the same as for the Linux example.

```
User Access Verification

Password:

3750>enable
Password:
3750#config t
Enter configuration commands, one per line. End with CNTL/Z.
3750(config)#
```

The **normal0** prompt is the device name `3750` followed by a greater than sign `>`. If we submit the `enable` command to change to enable mode on the device, the prompt changes to `3750#`. If we go into configuration mode, the prompt changes yet again to `3750(config)#`.

Tips

- Devices typically have a login and password prompt, but in some cases they may not. The prompts may vary from device to device.
- Prompts appear on a line by themselves and are usually left-justified.
- Prompts often contain device-specific or user-specific information (like device name or user name).
- Prompts can change when a device is put into a different mode.

About inheriting prompt definitions

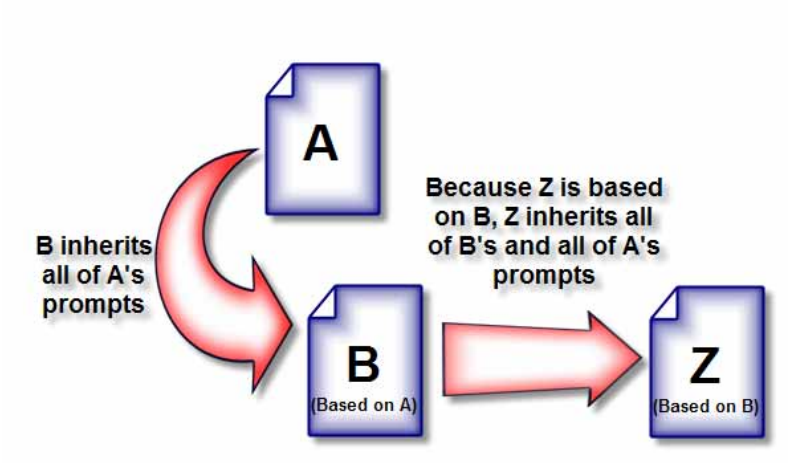
Because you typically add a new prompt definitions to the existing set of definitions for a session profile or testbed device, you usually choose to *inherit* the existing prompt definitions into the session profile and perhaps add to or delete from the inherited prompt definitions. You can use any of the following methods (detailed instructions follow):

- Inherit prompt definitions from the **Reference** session profile that you specified. The prompts from the reference profile are added to this profile. (This is the default behavior when you base one session profile on another.)

  Note Inheritance can nest as deeply as needed.

  - If session profile B is based on profile A (and therefore inherits A's property settings),
  - and profile Z is based on B,

- then Z inherits property settings from both A and B.



- Inherit prompt definitions and add new prompt definitions to the list of inherited prompts. If you decide to add prompts, then the inherited prompts become a part of the current session profile, but are not shown on the **Prompts** page in the list of prompts for the current profile.

  The inherited prompts are updated when the definitions in the reference session profile change.

- Start with the prompt definitions from the reference session profile and edit/delete them as needed. In addition, you can define new prompts for the session profile.

  Because the inherited prompts are now a part of the new profile, they are not updated when the prompts in the reference session profile change.

To inherit prompt definitions from the reference session profile

This is the default behavior.

1. Leave the **Include inherited prompts** check box checked (this is the default setting).
2. Leave the **Include additional prompts from list** check box unchecked (this is the default setting).

   The **Prompts** list does not display the inherited prompts. You cannot edit the property settings for any prompt definition.

To start with the inherited prompts and add new prompts

Note The inherited prompts are updated when the definitions in the reference session profile change.

1. Leave the **Include inherited prompts** check box checked (this is the default setting).
2. Check the **Include additional prompts from list** check box. The inherited prompts disappear from the **Prompts** list, but they are a still a part of the current session profile.
3. To add a new prompt definition, click **Add** . Set the prompt properties as described in the **Prompt properties** table. A **more** prompt property setting appears in **Terminal > Replay >**

Step Defaults > More. Another prompt property setting appears in **Terminal > Replay > Step Defaults > Completion**.

To start with the inherited prompts, edit them, and add prompts as needed

1. Starting from the default setting (checked), uncheck the **Include inherited prompts** check box.

   The **Prompts** list displays all inherited prompts.

2. To edit a prompt definition, select it in the list and then change settings as described in the **Prompt properties** table.

3. To add a new prompt definition, click **Add**  . Set the prompt properties as described in the **Prompt properties** table. A **more** prompt property setting appears in **Terminal > Replay > Step Defaults > More.** Another prompt property setting appears in **Terminal > Replay > Step Defaults > Completion**.

Note Because the formerly inherited prompts are now a part of the new profile, they are not updated when the prompts in the reference session profile change.

Tips for working with prompts

- ### What to do when iTest waits “forever” for a prompt during automated execution

  During replay or automated execution, a session can return a prompt with a format that iTest does not expect (for example, the prompt might include an timestamp or error message or might reproduce a failed command as a hint for correcting the command text). In this situation, iTest waits “forever” for a defined prompt that will never come.

  To enable the steps to replay without interruption in the future, you can “teach” iTest about the new prompt format by adding a prompt definition. iTest will recognize the prompt from then on.

  See “Learning prompts during automated execution” on page 464 for details.

- ### Connecting to a terminal server session where there is no prompt upon opening the session

  If you open a Telnet session through a terminal server to a console input, then there will be no prompt. While testing manually, you submit a carriage return (CR) by entering an empty command after the session is opened.

  During execution, iTest uses a prompt to know when to submit the next command, but if there is no prompt at the start of the session, then the CR step will timeout.

  Follow this procedure to avoid the problem:

  1. In the Test Case editor, select the **open** step for the Telnet session.
  2. In the **Step Properties** section, click **Telnet Step Defaults > Completion**.
  3. Set **Completion Criteria** to **Idle**. This forces the step to end if there is too much time without activity and then starts the next step starts immediately (to send out the CR).

Preventing errors when a prompt is delayed or is not recognized

Note The following tip applies to any CLI session type (for example, Telnet, Serial, Command Prompt, SSH, and so on). After you have read the tip, you can find further details in the "**Terminal > Replay > Step Defaults > Completion**" section in the chapter on the particular session type.

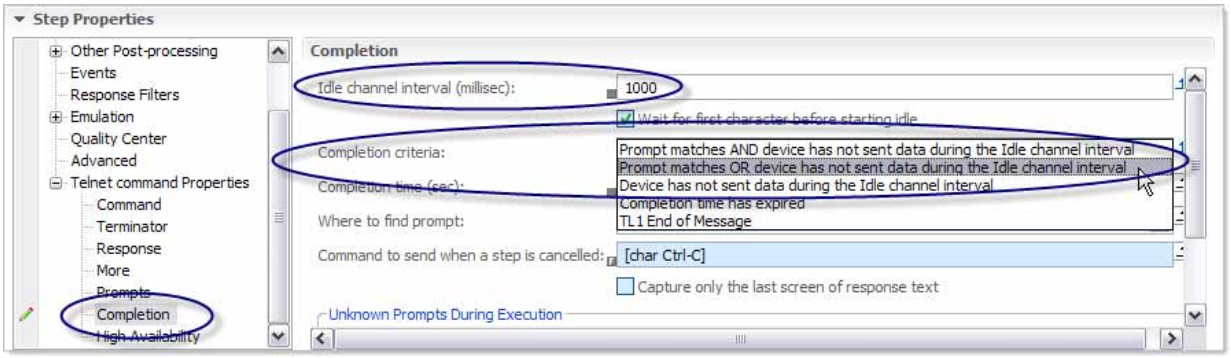
Some commands return very large responses. In some cases, due to network delay or delay at the device, it takes so long for the response to arrive and the prompt to appear that the **Completion time** that is configured for the step is exceeded.

Here is another condition that can result in an error associated with prompts: Some sessions can return a variety of prompts. If Spirent iTest is not configured to recognize the prompt that was returned for a step, the step can time out while waiting for a known prompt (either a built-in prompt or a prompt that you have defined).

To avoid both of these issues, you can configure settings that increase the time to wait for a known prompt and are more forgiving when unknown prompts are returned.

To enable the test case to tolerate late or unknown prompts:

1 Select the step. In the **Step Properties** section, go to <**Telnet** or **SSH** or other**> command Properties > Completion**.



2 As you know, iTest identifies possible prompts by noticing when the session returns text and then goes silent for a significant period of time (this time period is called the **Idle Channel Interval**). Increase the **Idle Channel Interval** to a value that is high enough to avoid errors, but not so high as to cause slow execution when there is a legitimate problem.

3 Select **Prompt matches OR device has not sent data during idle channel interval.**

The setting results in the following processing order: The step is completed once one of the defined prompts is received. If none of the defined prompts is received or no prompt is defined (the situation that we are trying to allow for), then the system waits for the specified **Idle Channel Interval** time, completes the step, and then continues to the next step.

CHAPTER 23

Virtual Testbeds (VTB): Testing with Emulated Sessions

Using emulation to provide a Virtual Testbed (VTB)

You can have iTest emulate ("pretend to be") any step or session. When you activate emulation for a step, iTest does not send the command to the session as usual; instead, iTest directly returns the response that you specify — the emulated response (typically, the response that the session returned the last time the test case executed. You can edit the response text as needed. No interaction with a session or device occurs — iTest returns the 'canned' response that you specify.

You can customize the behavior by editing the emulated response before executing the test case. For example, if the response text for the **show version** command for the last execution was "`Version 3.4`", and you need to develop the test case for the next version before you get the device software update, you can activate emulation for the step and change the response text to "`Version 3.5`" rather than "`Version 3.4`".

Typical uses of emulation

- Develop and troubleshoot a test case even when the DUT is not available
- Test a response map or analysis rule with a response that is difficult to get from a DUT
- Test variations in device behavior that are time-consuming to elicit from the actual DUT
- Write and verify test cases for new DUT commands that are still being developed, that is, start testing a feature before it is available
- If you have more testers than equipment, emulation enables you to automate tests without purchasing more equipment

Overview: Using emulation

You can emulate any session step that returns a response and any **call** or **run** step.

Note This is an overview — detailed instructions appears in "Emulating sessions in test case steps" on page 477.

Configuring the source of the emulated responses

Often, the emulated response is simply the response that was returned in a previous execution of the test case. There are other options:

- You can type any text into the Response view and the test case will use it as if it was returned from a device.
- You can have iTest take the response from a sample response in a specified response map.
- You can specify a response map library and have iTest use **Applicability** and **Priority** settings to auto-determine the response map and sample response to use.

Controlling emulation

To cause iTest to return an emulated response for any particular step, you must set the following property settings:

- Emulation must be ***enabled*** for the test case (emulation must be "allowed") — this is not the default setting.
- Emulation must be ***activated*** for the step (emulation must be explicitly "turned on" for the step) — again, this is not the default setting.
- Optional, but typical: You specify the text of the emulated response (you can tell iTest to use the response that was returned during the most recent execution against the actual device session). We discuss additional options in the following section.

Controlling emulation for devices in a topology

You can control how emulation is used for any device in a topology in any test case that uses the device. For details, see "Controlling emulation for devices in a topology" on page 484.

Fine-tuning emulation

The requirements for turning emulation on and some additional options give you significant freedom to design the test case to precisely meet your needs. You can:

- Emulate a single step, particular steps, all steps in a session, or all steps in a test case
- For a test case that has emulation enabled, temporarily disable emulation so you can run against the actual device sessions or topologies or testbeds. Later, re-enable emulation so you can continue developing without having to connect to sessions.
- Emulate all responses for a single session, particular specified sessions, or all sessions in a test case. You control emulation for all steps in a session by activating and deactivating emulation on session **open** steps.

Note When emulation is enabled on the **open** step, a warning message displays informing you that all other steps of this session will have emulation enabled automatically. That is, you cannot disable emulation for a single step, without changing the emulation for **open** step.

In addition, when steps have emulation enabled, disabling emulation at **open** step will *not* disable emulation at other steps.

- Use an option that executes steps at the same speed as in the actual test, at the fastest possible speed, or at a specified speed.
- To enable the iTest to dynamically determine the emulated response at runtime, field replacements are supported in many property settings.
- When you customize the structured data for an emulated response, iTest generates appropriate queries for the new response.

- You can use any of the following types of response content:
  - A response exactly as it was returned by the session in an earlier execution.
  - A response that you manually edit, optionally including field replacements that dynamically configure the response at runtime.
  - A response that you create “by hand” or that is generated by a script, optionally including field replacements.
- You can specify the source of the response for a particular step or session in the session profile or device for the step or in the step properties. By default, steps inherit the **External source** setting from the session profile or device. You can override the setting for a step.
- You have the option to specify the source of the response to be a sample response stored in a response map or response map library.

Emulating sessions in test case steps

Note These instructions describe the basic process of activating emulation. There are many powerful options for this feature — see “Emulation: Quick instructions” on page 483 for ideas.

Step 1: (Optional, but typical) Execute the test case

You can generate responses to use when emulating the test in either or both of the following ways:

- Populate all responses for the test case by executing the test case in the normal way (without configuring emulation). This provides iTest with the default responses to use when emulating sessions. You can use these responses as they are or edit them to suit your needs.

Important If you will use a sample response from a response map or response map library to provide the emulated response, then you must first create the response map or add the sample response to an existing response map. See “Creating a response map: Instructions” on page 554 and “Adding a sample response to an existing response map” on page 557.

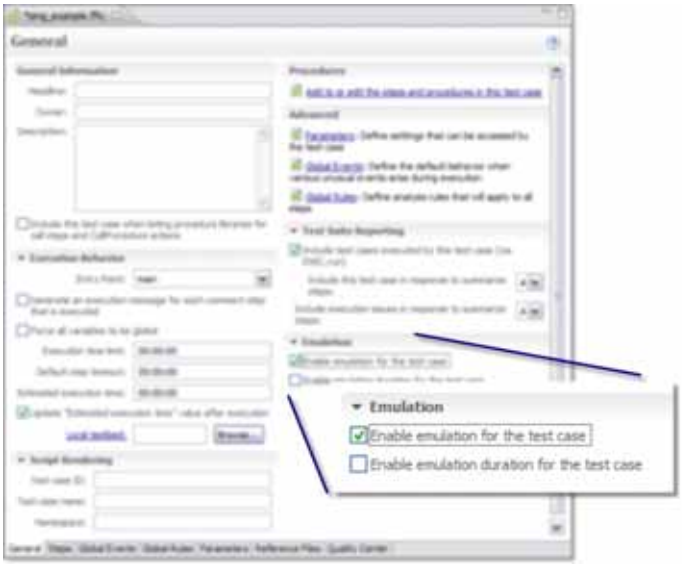
- Create a response manually by typing or pasting it into the Response view for a step.

Step 2: Enable emulation for the test case

When emulation is enabled for the test case, then any step for which emulation is activated will receive an emulated response. Use either of the following methods to enable emulation for the test case:

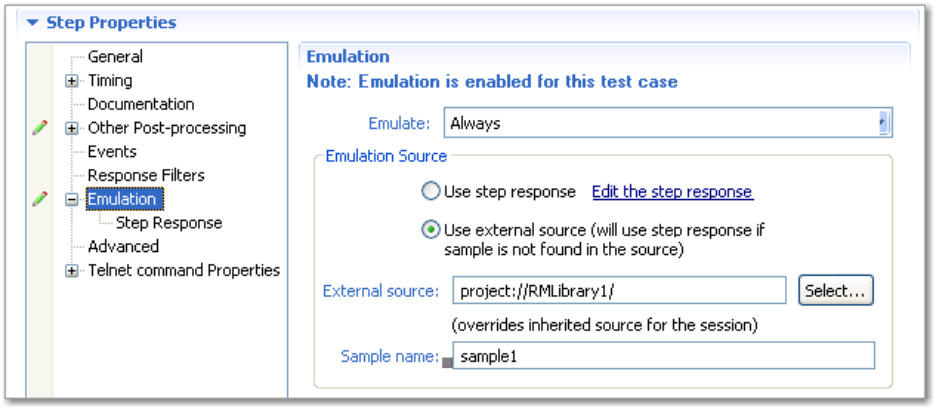
- In the **Test Case** menu, select **Emulator > Enable Emulation for the Test Case**
- On the **General** page of the Test Case editor: In the **Emulation** section:
  - Check **Enable emulation for the test case**

- Optional. Check **Enable emulation duration for the test case** to cause an emulated step to execute in the same amount of time as the actual step. Uncheck to execute emulated steps as fast as possible (default).



Step 3: Activate emulation for particular steps or sessions and specify the source of the emulated response

1. Select one step or multiple steps whose responses should be emulated.
   - You can emulate any **call** or **run** step and any step in a session.
   - To emulate all steps in a particular session, select the **open** step for the session and continue with these instructions.
2. On the **Step Properties > Emulation** page, set the **Emulate** property to **Always.**



Emulate	**No**: This is the default setting: Do not emulate the response; send the command to the session in the typical way. **IMPORTANT** If the **open** step for a session is emulated, then all steps in the session are emulated, including steps for which **Emulate** is set to **No**. **Always**: Rather than interact with the session, return the response that is specified by the **Emulation Source** property group.

3 Specify the **Emulation Source** (where to get the emulated response):

- **Use step response** causes iTest to use a response that had been returned for an earlier execution of the step (or an edited version of such a response). If you select this option, then click the **Edit the step response** link to configure the step response. Skip to "How the sample is determined at runtime" on page 480 for further instructions.
- **Use external source** causes iTest to emulate the response using a sample response from a response map or response map library. (To use a sample response as the emulated response, you must have previously configured the sample while defining a response map, as described in Step 3 in "Configuring a sample response" on page 563.)

  By default, the **External source** property is blank and the setting is inherited from the **External source** setting in the session profile or device associated with the step. To override that setting, you specify a setting here, in the **External source** property for the step. Follow these steps to specify the external source:

a Click **Select**. In the **Specify External source** dialog box, specify one of the following:

- The **Use the source configured for the session** setting is the default when the **Use external source** check box is not checked. That is, use the setting that is inherited from the **External source** setting in the session profile or device associated with the step. If no source is specified there, then use the response that is specified by the **Emulation > Step Response** property group for the step.
- Default: Use the specified **response map library** (specify either a library project or the catalog for a library)
- Use the specified **response map**

Tip We recommend that you specify a response map library rather than an particular response map as the **External source**. In this way, rather than requiring you to specify a response map for each step in a test case, you can take advantage of iTest's ability to auto-select the appropriate map based on **Applicability** and **Priority** property settings.

---

b Set the **Sample name** property: If you use the **External source** property to specify a response map library or response map as the external source for the emulated response, then use the **Sample name** property to specify the name of the response sample to use (that is, the sample from within the appropriate response map).

To enable the test case to dynamically determine the sample response at runtime, field replacements are supported in this field.

How the sample is determined at runtime

- If the **External source** is a response map library, then:
  - iTest first auto-selects a response map based on **Applicability** and **Priority** property settings. iTest then selects the particular response from within the response map based on the setting of the **Sample name** property. If no **Sample name** value is specified (the field is blank), then iTest uses the first sample response that appears in the appropriate response map.

    The response map chaining feature enables you to specify that, during the search for the sample response to use for emulation, any step that does not find an applicable map in the specified response map library should also check for applicable maps in one or more other libraries. See "Making use of existing response map libraries: Chaining response maps" on page 559.
  - If iTest cannot identify an appropriate response map based on **Applicability** and **Priority** property settings, then iTest uses the step response (as specified by the **Emulation > Step Response** property group for the step).
- If the **External source** is a response map, then iTest selects the particular sample response from within the response map based on the setting of the **Sample name** property. If no **Sample name** value is specified, then iTest uses the first sample response that appears in the appropriate response map.
- For both the response map library and response map options: If the **Sample name** is specified, but no sample with that name is found, then iTest performs the following actions:
  - Return no response for the step
  - Generate an **onEmulationSampleNameNotFound** event
  - Set the test case result to **Fail**
- If the **External source** property is empty and no setting is specified in the session profile or device associated with the step, then iTest uses the step response (as specified by the **Emulation > Step Response** property group for the step).

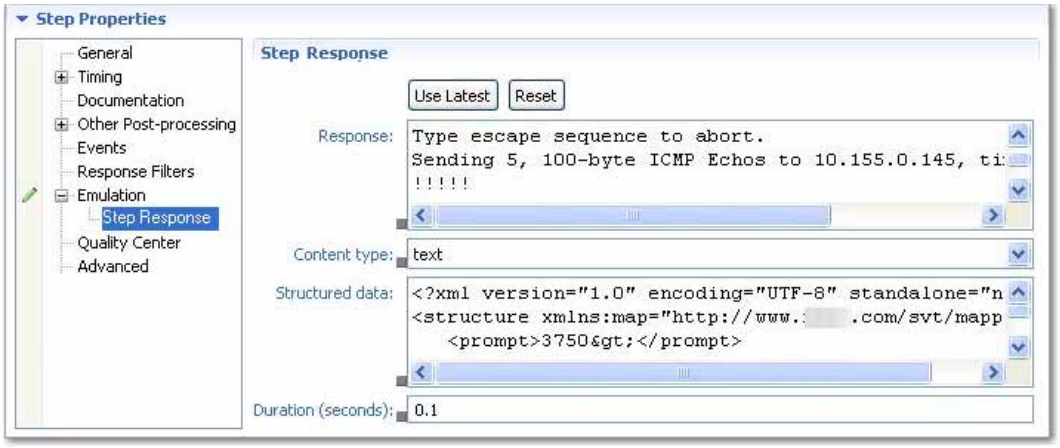
Step 4: Configure the "step response" (the emulated response for the step)

iTest uses the response that you configure in this step in the following cases:

- The **Use step response** option is selected
- The **Use external source** option is selected, but the **External source** is not specified for the step or in the session profile or device associated with the step

- The **Use external source** option is selected and the **External source** is a response map library, but iTest cannot identify an appropriate response map based on **Applicability** and **Priority** property settings.

On the **Emulation > Step Response** page for the step, configure the following properties. Remember that iTest can perform runtime substitution for any properties marked by the field replacements indicator .



Use Latest	Click the button to use the data that the session returned the last time the test case executed. This populates the **Response**, **Structured data**, and **Duration** properties and copies application-specific queries from the last response. The queries are not shown in the emulation properties and cannot be edited, but they will appear in the Query view after a step has been emulated
Reset	Click the button to clear the property settings.that define the emulated response: **Response, Content type, Structured data**, **Duration** and application-specific queries**.**
Response	This multi-line text box holds the text that will be returned for an emulated step. To populate the property with the text of the response returned during the most recent execution, click **Use Latest**. You can modify the text to meet your needs. For example, if the response text for the **show version** command for the last execution was “Version 3.4”, and you need to develop the test case for the next version before you get the device software update, you can change the response text to “Version 3.5”.
Content type	Specify the format of the response data. This setting ensures that structured responses like XML and TL1 are correctly parsed. **Note** For more information, see “Configuring sessions and test case steps for TL1 devices” on page 1215. Default: text

Structured data	This multi-line text box holds the structured part of the response that will be returned for an emulated step. To populate the property with the structured data returned during the most recent execution, click **Use Latest**. As with the **Response** text, you can modify the text to meet your needs **Note** iTest emulates only the structured data from the response; it does not emulate structured data that is appended by response mapping. The **emulated/source** element in the structured data identifies where the emulated response actually came from: • The value "step" means the response came from the **Step Response** properties • A value of "<source_uri>#<sample_name>" appears when the response came from a sample in a response map
Duration	This property is used only if you check **Enable emulation duration for the test case** on the **General** page of the Test Case editor. This property specifies how long it will take to execute the emulated step. To populate the property with the time of the latest actual response, click **Use Latest**. You can modify the setting to change the duration of emulated steps. For example, you can speed up or slow down a **reboot** step. Specify 0.0 to execute as fast as possible. Default: 0.0

- **Step 5: Emulation is now active for the specified steps**

  When you execute the test case, the resulting test report indicates the steps that used emulated responses. See "Setting preferences for emulation" on page 487.

Adding a sample response (for use as an external emulation source) to an existing response map document

When you specify an external emulation source for an emulated response, iTest uses a sample response that is stored in a response map document. See "Adding additional samples to a response map document" on page 566 for instructions on adding a sample response to a map for this purpose.

Emulation: Quick instructions

To:	Do this:
Create a response "from scratch" when there is not yet anything to test	Select the step and then type the text into the Response view.
Enable emulation for the test case so that any step for which emulation is activated will execute with the emulated response	Either one of the following: • In the **Test Case** menu, click **Emulator > Enable Emulation for the Test Case** • On the Test Case editor **General** page, check **Enable emulation for the test case**
Turn off emulation for the test case so you can run a test against the actual testbed devices	Either one of the following: • In the **Test Case** menu, click **Emulator > Disable Emulation for the Test Case** • On the Test Case editor **General** page, uncheck **Enable emulation for the test case**
Emulate all responses for a particular session, regardless of whether any particular step has activated emulation	1. Select the **open** step for the session. 2. On the **Emulation** properties page, for the **Emulate** property, select **Always**. 3. Repeat as needed for any other sessions.
Using an existing response map library to provide emulated responses for all steps in a session (some steps might currently use the response map for analysis)	in the session profile or device for the session, set the **External source** property to the response map library and then enable emulation for the session's **open** step. **Note** If you had already specified particular response text on the **Emulation > Step Response** page for some steps, then you can easily reset all emulation settings to make use of the external source (in particular, checking the **Use External source** checkbox for all steps): In the **Emulation** menu, select **Clear Emulation For All Steps**.
Remove all emulation from the test case (in the case, for example, that the test case is fully debugged and ready for use)	In the **Test Case** menu, click **Emulator > Clear All Emulation Properties for the Test Case** **Note** If you want to keep a version of the test case that uses emulation, save a copy of the test case before performing this operation.
Update all steps that use emulation to use the responses from the latest execution against actual testbed devices	In the **Test Case** menu, click **Emulator > Update Emulation Responses for All Steps**
Activate emulation for all sessions in the test case or for most sessions — all session responses are emulated	In the **Test Case** menu, click **Emulator > Activate Emulation for All 'open' Steps**. This sets the **Emulate** property to **Always** for all open steps and also enables emulation for the test case. To emulate responses for only particular sessions, click the menu item and then, for each session that should not be emulated: 1. Select the **open** step. 2. On the **Emulation** properties page, for the **Emulate** property, select **No**. 3. Repeat as needed for any session.

Disable emulation for all sessions in the test case — steps (other than 'open' steps) that have emulation turned on will still be emulated	In the **Test Case** menu, click **Emulator > Deactivate Emulation for All 'open' Steps**. This sets the **Emulate** property to **No** for all open steps. You can return to emulating all sessions by clicking **Emulator > Activate Emulation for All 'open' Steps**
Execute the emulated test case as fast as possible (ignore the duration for execution against actual testbed devices)	On the Test Case editor **General** page, uncheck **Enable emulation duration for the test case**
Execute the emulated test case at the same speed as a execution against actual testbed devices	On the Test Case editor **General** page, check **Enable emulation duration for the test case** During execution, iTest executes all emulated steps in the same amount of time as it took for the step in the actual session. As a reference, iTest uses the execution that you specified as the basis for the settings in the Emulation properties group. You can modify the **Duration** setting for any particular step as needed.

Controlling emulation for devices in a topology

To Control emulation for all devices in a topology

This method is useful for controlling emulation in any test case that uses the topology with a single setting.

To control emulation settings for all devices In the topology while working in the **Topology** editor:

- On the main menu, click **Topology > Emulation**

  The following three options are applied to all devices in the topology regardless of any current device setting:

  Emulate All Device Responses: Regardless of the setting in the test case, use emulated responses if available (devices are shaded in teal) 

  Do not Emulate Any Device Responses: Regardless of the setting for the topology or for any test case, do not use emulated responses (devices are shaded in lilac) 

  Use Test Case Emulation Settings: Use the settings that are currently configured in any test case that uses any device in the topology.

  The following two options control whether the emulation settings that might be defined for individual devices are used or not:

  Enable Device Emulation Settings: Use the emulation settings (if any) defined for each device in the topology

  Disable Device Emulation Settings: Do not use the emulation settings for any device in the topology (devices are shaded in lilac) 

- Right-click in a blank area of the canvas and select one of the following options (as described earlier):

  Enable Device Emulation Settings (default)

  Disable Device Emulation Settings

To control emulation for a particular device in a topology

This method is useful for controlling emulation in any test case that uses the device with a single setting.

In the **Topology** editor, right-click the device and select one of the following options:

Use Test Case Emulation Settings: (default) Use the settings that are currently configured in any test case that uses the device.

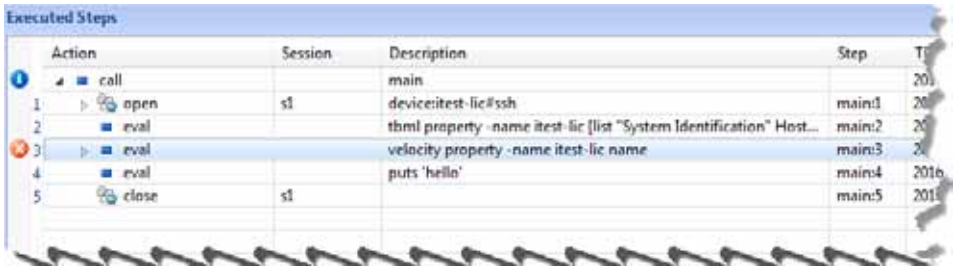
Always: Regardless of the setting for the topology or for any test case that uses the device, use emulated responses if available. (the device is shaded in teal) 

Never: Regardless of the setting for the topology or for any test case that uses the device, do not use the emulated responses. (the device is shaded in lilac) 

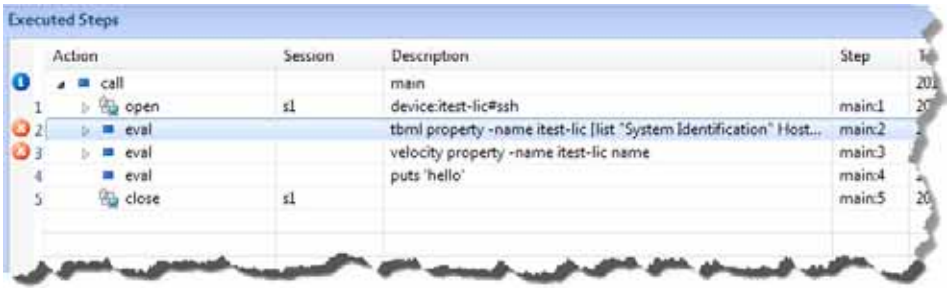
Reservation of emulated test cases

The following applies during reservation of emulated test case depending on the emulation preferences settings (see “Setting preferences for emulation” on page 487).

- When **Enable emulation for the test case when it is enabled for a step** is *enabled* in a test case (page 487):
  - The eval step with "tbml" command will use a local topology for test execution (even without a reservation).
  - The "eval" step with "velocity" command will display an execution error if a reservation does not exist.

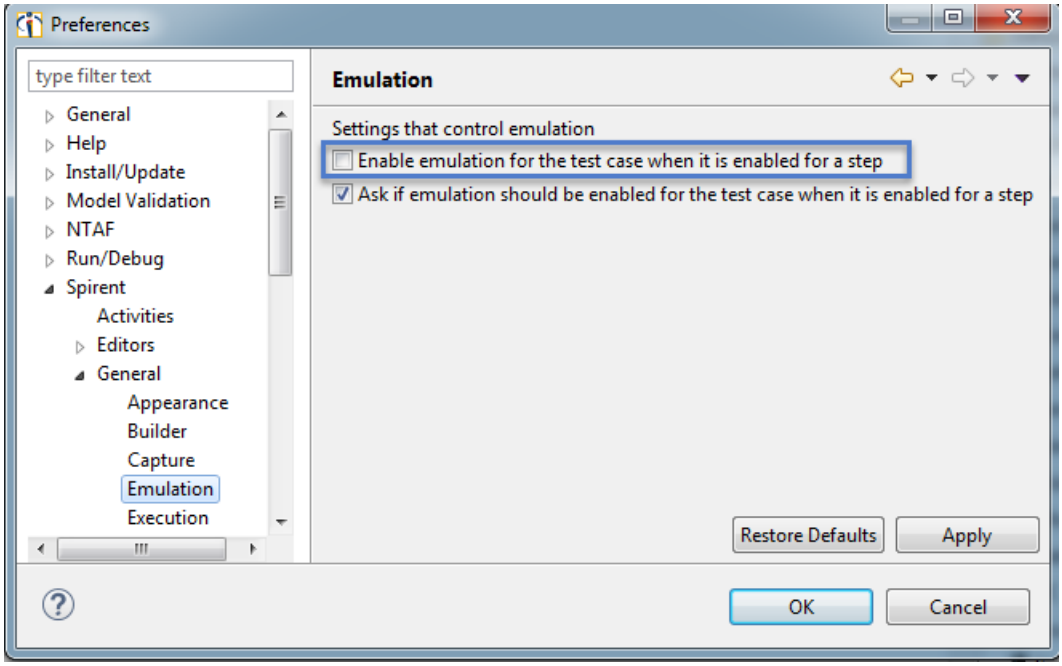


- When **Enable emulation for the test case when it is enabled for a step** is *not enabled* in a test case (page 487):
  - The "eval" step with "tbml" command does not use the local topology for test execution will display an execution error if a reservation does not exist.
  - The "eval" step with "velocity" command will display an execution error if a reservation does not exist.



Setting preferences for emulation

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > General > Emulation.**



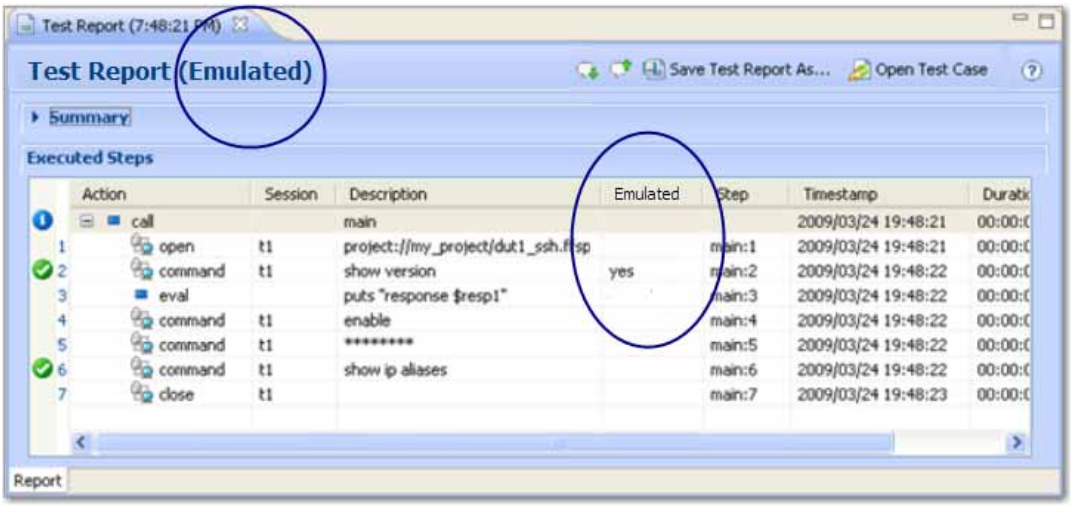
General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Enable emulation for the test case when it is enabled for a step	Check the box if you find that you want emulation enabled for all steps in a test case when you start working on an emulated step. **Note** This property is purely a convenience — you can achieve the same functionality by checking **Enable step emulation** on the Test Case editor **General** page or by selecting **Enable step emulation** in the **Test Case** menu. See the **Ask if emulation should be enabled** property. Default: unchecked
Ask if emulation should be enabled for the test case when it is enabled for a step	If you check the **Enable emulation for the test case when it is enabled for a step** property, then you can check this property to ensure that you are prompted whether or not to enable emulation for all steps in the test case. Default: checked

Test reports for emulated executions

Whenever any step in an execution was emulated, the test report changes as follows:

- The title of the page is **Test Report (Emulated)**
- An **Emulated** column appears after the **Description** column, indicating **yes** for any step whose response was emulated.



CHAPTER 24

iTest Topology editor

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Overview: iTest Topologies

You use the Topology editor to define a ***topology*** document: all of the physical devices, cards and interfaces on the devices, links between devices, and session configurations defined for devices.

A topology document is a collection of the information that test cases need to access the devices and sessions (for example, the IP address of the device and login information). Each session configuration is based on a default iTest session type (such as Telnet, Web, SNMP, and so on) or on a session profile. Third-party providers can supply other session types.

When you associate a topology with a test case, then all of the devices in the topology are available to start sessions in the test case. To run a test case on a different set of physical devices (that is, a different topology), you update the test case to refer to a different topology document.

When you connect to Velocity and publish a topology, all of the resources defined in the topology become available for reservations in Velocity.

File format of iTest topology documents

- iTest creates TBML-format files (file extension **.tbml**)
- You can open an iTest topology document in any TBML-compliant editor
- The iTest Topology editor can open, edit, and save any TBML-compliant file

Commands that return information about topologies

The group of commands returns information about devices, cards, ports, and inks in a topology or about the topology itself. See “Commands that return information about topologies” on page 524.

Overview: Creating a topology

The process of defining a topology in iTest is quite flexible — there is no required order for defining devices or links in a topology, and you do not need to fully define devices or links in order to define relationships between devices.

You can define a topology at many different levels of detail, depending on your needs:

- If your organization uses several physical topologies, each of which is an instance of a "typical" or "base" arrangement of devices and links, you might consider defining an ***abstract*** topology:
  - Devices do not need to have session profiles defined
  - If there are session profiles defined, you do not need to configure all session properties
  - You can use parameters to represent the runtime values

  For example, you can define a Telnet session profile that specifies the device's IP address or hostname using a parameter.
- A topology that will be used to supply devices and session configurations for automated test cases requires the most detail.
  - Each device must include at least one session configuration
  - Required session properties must have either hard-coded or parameter values

Once you have defined the topology, a test case developer can open the **General** page on the Test Case editor and specify the **Local topology** for the test case. As a result,

- For **open** steps in the test case, the test case developer can easily select any of the devices in the topology from a drop-down list
- Use the Command wizard that takes you through creation of all of the commands. See "TBML Command Wizard" on page 502 for details
- Parameters defined in topology session profiles can be accessed by any step in a session using the "TBML Command Wizard".

Note You may also easily convert an existing Testbed to Topology. See "Migrate Testbed to Topology" on page 506.

Why it is a good idea to define topologies

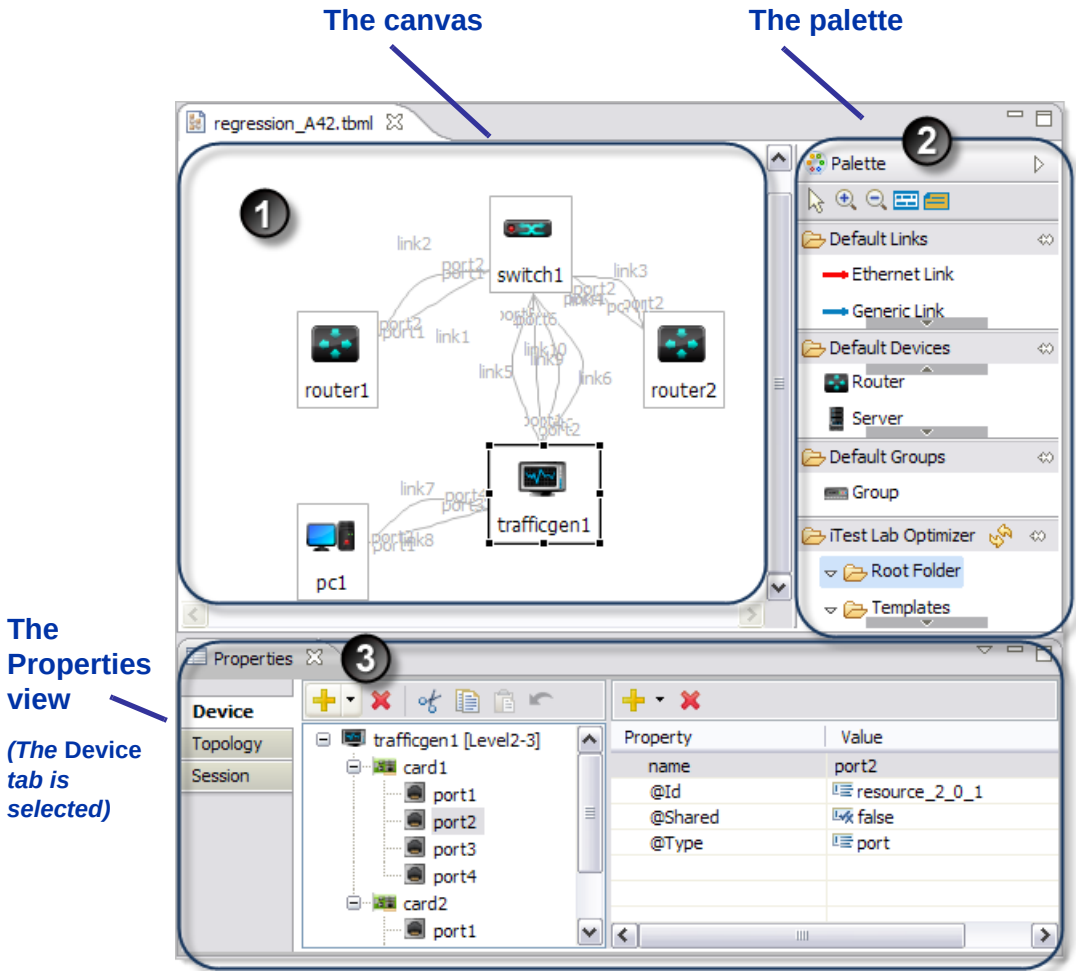
Topologies are powerful in many ways:

- When you connect to Velocity and publish a topology, all of the resources defined in the topology become available for reservations in Velocity.
- Once you have defined the topology, a test case developer can open the **General** page on the Test Case editor and specify the **Local topology** for the test case. As a result, when they add an **open** step to the test case, the list of available sessions on devices that they see is the list that you specified for the topology — no guesswork on the test developer's part.
- When you define a topology device by basing it on a default session type, you avoid cluttering your system with many session profile documents that might differ only in small details. Easier to maintain and to understand.

- You can design your test cases to be easily run against a variety of topologies (for example, with devices that differ only in IP address and software version, and so on). As a result, you can run a test against several different sets of physical devices by changing a single setting; the **Local topology** used by the test case.
- There is another option for identifying the topology that any test case should use when the test case does not specify a topology; the Global topology (described in "Global topology" on page 539).

Layout of the Topology editor

Here is a simple topology open in the Topology editor. Most Topology editor tasks are easy to perform. Read this section first and then see "Topologies: Quick instructions" on page 493 for the most common tasks.



The canvas is where you place elements of the topology

The canvas displays the devices and the links between devices that make up the topology. In addition, you can display notes and labels directly on the canvas to further document the topology.

Tip Many of the most common tasks appear in the context menu when you right-click an item on the canvas or in the **Properties** view.

The information icon **i** on a device indicates that non-inherited values are specified in one or more of the session profiles or properties defined for the device.

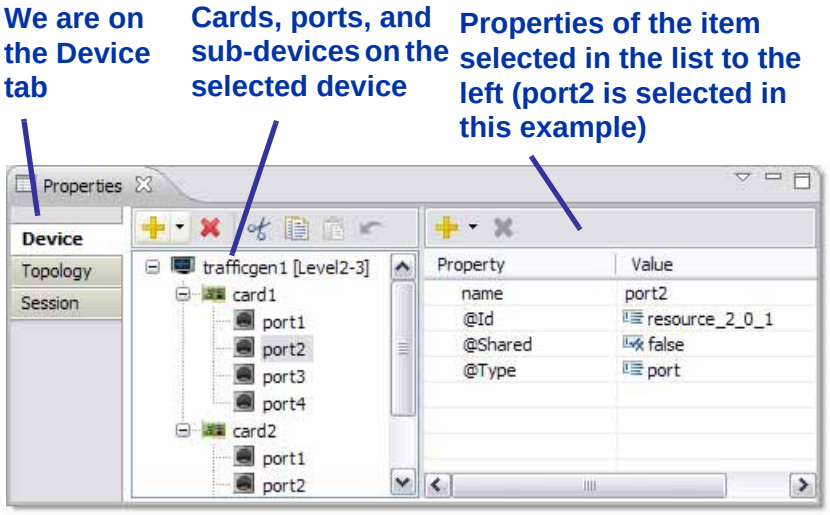
2 *The palette provides the tools you need to add devices, links, labels, and notes*

- Drag-and-drop a device type or **Link** from the palette onto the canvas.
- The palette also provides a **Zoom** tool and drag-and-drop **Note** and **Label** tools.
- The **Group** tool enables you to visually group devices into a logical grouping. A group is meant to improve understanding of the organization of the topology and does not affect the function of any topology element.
- The **Velocity** section displays the folder structure of the topologies and resources that are defined on the Spirent iTest server. iTest periodically downloads any data updates from the Spirent iTest server to sync the data in the iTest workspace (for example, the definition of a new resource).

3 *The Properties view*

The Properties view takes on special functions while you work in the Topology editor. When you select a device, a link, or even the canvas, the view displays tabs that enable you to configure the selected item.

For example, to change a property of one of the ports on a device: Select the device on the canvas, (in the example canvas, the dark outline on **trafficgen1** indicates that it is selected), and then click the **Device** tab. Select the port in the list under the device (**port2** in the example) and then change the value of the property as needed.



- Use the **Device** tab to add a card to the device, to add a port to the device or to a card, or to add a generic sub-device to the device. In addition, you can specify property and attribute values for devices, cards, and ports. See "Topology editor: Properties view, Device tab" on page 514.
- Use the **Topology** tab to document the topology's design history for the benefit of coworkers, as described in "Topology editor: Properties view, Topology tab" on page 512.

- Use the **Session** tab to add, view, and edit session profiles for devices, as described in "Topology editor: Properties view, Session tab" on page 513.
- Use the **Link** tab to specify property and attribute values for the selected link (connection). For example, to rename a link, change the value of the **name** property.
- Use the **Ruler and Grid** tab to specify how new devices should align when added to the canvas, whether to display the ruler, and to specify the appearance and operation of the alignment grid for the canvas. See "Topology editor: Properties view, Ruler and Grid tab" on page 518.

Topologies: Quick instructions

Many of the operations in the following table are described in detail elsewhere in this chapter, but the table may give you all you need to create and manage topologies. Be sure that you are familiar with the "Layout of the Topology editor" on page 491 before you use the instructions.

Tip Remember that many of the most common tasks appear in the context menu when you right-click an item on the canvas or in the **Properties** view.

The tabs that you use to manage the topology definition are described in:

Topology editor: Properties view, Topology tab See page 512.

Topology editor: Properties view, Session tab See page 513.

Topology editor: Properties view, Device tab See page 514.

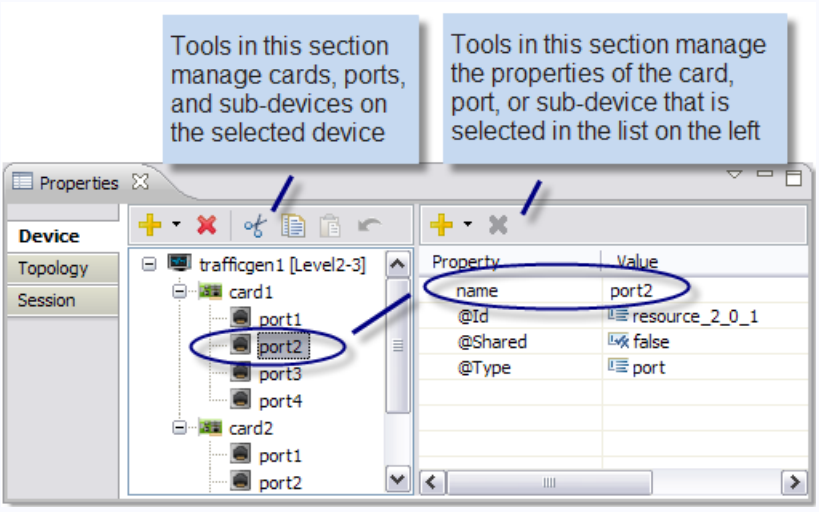
Topology editor: Properties view, Link tab See page 518.

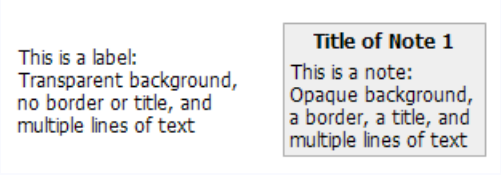
Topology editor: Properties view, Ruler and Grid tab See page 518.

To:	Do this:
Define a topology	Click **File > New > Topology** If you move a topology into the Favorites view, the value of the **@Name** attribute appears in the **Headline** column.
Open a topology document in the Topology editor	While working in **iTest Activities**, click **Build a topology**. While working in any other perspective: In the Favorites view or the iTest Explorer, right-click a **t**opology document and then click **Open.**
Open the Properties view while working on a topology	Click anywhere on the canvas. Alternatively, right-click anywhere on the canvas and then select **Show Properties View**

Add a device to a topology	1. Select a **Device** type in the palette and then either click on the canvas or drag to the desired location. 2. The device is given a unique default name. **Tip** The name that appears on the canvas for a device is the **name** property value. To change the **name**, select the device on the canvas, click its name, and then type a new name. Alternatively, select the **Device** tab in the Properties view and type a new value for the **name** property. The **name** appears in the **Select a Session Profile or Device** dialog box when a test case developer is specifying the device for an **open** step. 3. Use the grab handles to resize the device graphic as needed.
Add a device to a topology	You can add a special kind of "device" to a logical topology – VLAN, and link resources' ports (real or abstract) with it. iTest assigns the VLAN network ID. • VLAN must not have ports or a **condition** property • A link with a VLAN as one of the endpoints is called L2 link or VLAN link. A link cannot have a VLAN at both endpoints. • A link with both endpoints being real or abstract devices' ports (not VLANs) can be explicitly marked as VLAN link. In this case, this link is treated as if there is an implicit VLAN device between the endpoints. This is done to have a convenient way to create VLANs with two devices. • Other links are called L1 links. • L2 links must visually differ from L1 links. May be, dashed line, or something like this. • A resource may be connected to multiple VLANs, if it has multiple ports. • VLAN links are always bidirectional.

Add and configure a link between two devices **Add a link from a device to itself**	**Note** The instructions in this table describe adding a single link between ports. To add multiple links quickly, use the **Create Connection** wizard, as described in "To quickly add multiple links (connections between ports)" on page 501. 1. In the palette, select a link type (for example, **Ethernet Link**). All links that you now add will be that type until you select a different type. 2. Now add the link: • To add a link from a device to itself (the device is both source and target). Click the device. • To add a link from one device to another: Click the source device and drag to the target device. • Alternatively, when you hover over or click a device, link "handles" appear. The handle with an outgoing arrow represents an outgoing link and the other represents incoming (in the example, we selected the outgoing link, that is, **router_1** is the source device). Drag a handle to the target or source device to create a link of the type currently selected in the palette.  3. The editor gives the new link a unique default name and connects it to the next unconnected port on the source and on the target (if needed, the editor adds a new port). The names of the link endpoints (ports) appear on the canvas. • To modify the links, click the **Source** or **Target** property and select the desired card or port. • To take the new default port and add it into a new card on a device, add a new Card, add a new port to it, and then point the link to the new port.
Add a card, a port, or another device to a device **Add a port to a card**	When you add a link to a device, iTest adds a default port on the device and assigns the link to the port, as described in the preceding instruction. In addition, you can add cards, ports, or 'sub-devices" to a device or to existing cards, ports, or sub-devices on a device. • To add to the device: On the canvas, click the device and then, on the **Device** tab, click the arrow on the **Add** button . Select the item to add. (Clicking the part of the button adds a card.) • To add to an existing card, port, or device on a device: Select the item in the list on the **Device** tab and then click the arrow on the **Add** button . Select the item to add. The new item is given a default **displayName** that indicates its position in the hierarchy. For example, when new cards are added to r**outer1**, then the cards are named **card1**, **card2**, **card3**, and so on.

Configure cards, ports, or sub-devices on a device	• When you add a link to a device, the editor adds a default port on the device and assigns the link to the port. You decide whether or not to edit the port's property settings to match the actual configuration of the physical device. • Generic Devices (the kind of sub-device that you can add to a device) have no default **@Type** attribute value. You can set the attribute value as needed (for example, **workstation**), • To change a property of one of the cards, ports, or devices within the device, select the item in the list under the device (in the example, **port_5_2** on the **trafficgen_5** device) and then change the value as needed: .  **Note** The @ symbol denotes a attribute rather than a iTest property.
Add and configure a session profile for a topology device	See "Add, edit, or remove a session configuration for a topology device" on page 500.
Add a property to a topology, to a device, or to a link	You may want to add a property definition to the current default list of property definitions. For example, for the device with the default abstract (logical) **name** of **router1**, you might want to add the **displayName** property that will hold the friendly name of the actual physical device, like **4268__rtr_rev3.52**. Select the device or link. To select the topology, click anywhere in a blank area of the canvas. 1. In the appropriate tab on the **Properties** view, click . 2. In the **New Property** dialog box, specify the **Name** for the property and its **Value**. 3. Select the **Vendor** (Unique identifier of the provider of the session type templates — typically **com.fnfr** for Spirent.) **Note** The default properties and attributes are provided by the session configuration provider. Adding a property here does not add the property to the provider's definition.

Add a property collection to a topology, to a device, or to a link	Select the device or link. To select the topology, click anywhere in a blank area of the canvas. 1. In the appropriate tab on the Properties view, click the arrow on the **Add** button and select **Add Property Collection**. 2. In the **New Property Collection** dialog box, specify the **Name** for the property collection. 3. Select the **Vendor** (Unique identifier of the provider of the session type templates — typically **com.fnfr** for Spirent.) **Note** The default properties and attributes are provided by the session configuration provider. Adding a property here does not add the property to the provider's definition. 4. The editor adds the property collection. To add a property to the collection, select the collection and click
Place devices into an orderly arrangement on the canvas	Move the device by dragging. By default, devices snap to an underlying grid. To use the tools that control grid settings, click anywhere in the background of a topology. Click the **Rulers & Grid** tab in the Properties view and change the setting. To set the **Rulers & Grid** settings for all topologies displayed in the Topology editor, see "Set preferences for Topologies" on page 540.
Add a label or note anywhere on the canvas	You can add a label or note anywhere on the canvas.  **To add a label:** In the palette toolbar, click **Label** and then click at the desired location. Type the label text. Use the Enter key to add lines. **To add a note:** In the palette toolbar, click **Note** and then click at the desired location. Type the title text in the bold text box. Type the body text in the next text box. Use the Enter key to add lines.
Edit a device's session profile	1. Select the device. 2. On the Properties view, click the **Session** tab and then select the session profile in the list. 3. Click . The the **Edit Session Profile** page opens. 4. Edit properties as needed and then click **OK**.
Cut / Copy / Paste / Delete something in the topology	In the canvas or in the Properties view, select the items and then click one of or . **Delete** deletes all children. Drag the cursor to select. any number of objects You can paste into other topologies.
Start a session on a device	1. Select the device. 2. On the Properties view, click the **Session** tab and then select the session profile in the list. 3. Click .

Import a topology that was created on a different system	As long as the file is TBML-format, iTest can open it and you can edit and save it as a iTest topology. In addition, any TBML-compliant editor can open a iTest topology file. Use Ctrl+click and Shift+click or Ctrl+A for multi-select. You can copy/paste one or more devices or properties from one topology to another.
Save a topology as an image file	As a communication tool, you may want to share a graphical image of a topology or an item in the topology or a selected group of items. 1. On the canvas, right-click an item or the canvas background and select **File > Save As Image File**. 2. On the **Save As Image File** dialog box, specify the following values. **Folder**: Specify the location of the file in the file system. **File name**: Specify the name of the image file. **Image format**: Specify the image format of the file. To save the image as an HTML file, check the **Export to HTML** checkbox (the **Image format** property is then ignored). Check **Overwrite existing file without warning** to not display the dialog box that asks you whether to overwrite an existing file. 3. Click **OK**.
View resolved resources from abstract topology in iTest GUI	See "View resolved resources properties (from Abstract Topology)" on page 515.
Set preferences for Topology operations	See "Set preferences for Topologies" on page 540

Add and configure a topology

As described in "Overview: Creating a topology" on page 490, the process of defining a topology in iTest is quite flexible. For that reason, the following instructions are general in nature and you might follow a different course.

◆ To create a topology

1 Click **File > New > Topology**.

2 The **New Topology** wizard starts.

- **a** Specify the folder that will hold the topology (TBML) file either by typing the path or by selecting it in the project tree.
- **b** Specify the **File name** for the file.
- **c** If the topology should be added to Velocity, then check **Synchronize this topology with the Spirent iTest server**.

  This option ensures that if changes are made in Spirent iTest to the properties or ports for a resource, then all topologies that include the resource are auto-updated (synced).
- **d** Click **Finish**.

A new blank TBML-format file is created and opened in the Topology editor. In addition, the Properties view opens.

3 On the Properties view, the **Topology** tab is selected. Document the **Name** and **Description** of the topology, as described in "Topology editor: Properties view, Topology tab" on page 512.

4 Each line on the **Devices** tab defines a device. Add a device as described in "Topologies: Quick instructions" on page 493.

The name is very important. Here is an example that shows why: In this example, we will name a device **DUTRouter2** in both the **LocalTopology** topology (the document that defines your personal development topology) and the **SharedRegressionTopology** topology (that defines the QA regression topology) — only the IP addresses are different.

As a result, any test case that refers to **DUTRouter2** in a step can, without modification, use either of the two topologies (you have only to specify the appropriate topology on the Test Case editor **General** page). Once you finish developing the test case, you can hand it off for regression without any changes.

---

5 Add and configure resources and device sessions as needed.

To define an abstract resource, use any of the following methods:

- Drop an appropriate resource template onto the palette.

  The **condition** property is **template[<type>]** (for example, for **template[Server]**, the resource inherits **Server**)

  All ports are also abstract and contain a **condition** property.

- In an existing topology, convert an existing resource into an abstract resource by xxx.

See also "View resolved resources properties (from Abstract Topology)" on page 515/

6 Optional. If appropriate, configure how emulation is used for any device in a topology in any test case that uses the device. For details, see "Control emulation for devices in a topology" on page 512.

7 Save the topology document.

Add, edit, or remove a session configuration for a topology device

Important Session profiles that you define for devices in a topology are saved as part of the topology file and are not saved as separate session profile documents.

To define a session for a device, you use one of the following methods:

- **Session type**: Inherit all property settings from the default iTest template for a session type and then configure particular properties as needed.
- **Create reference session profile**: Select an existing or create a new session profile (typically a reference profile) document (.ffsp), from which the new session configuration will inherit property settings. You can change property settings as needed.

To define a session for a resource

1. On the canvas, right-click the resource and select **Add Session**.

   (Alternatively, select the device on the canvas. On the Properties view, click the **Session** tab and then click) The **Add Session Profile** page displays. Enter the details as below.

2. **Profile Name:** type the **Profile name** for this particular set of session configuration settings. iTest provides a default name that includes the session type.

3. **Language:** Select the language that will be used to create the session profile. You may use the default language displayed (as set in "Preferences: Spirent > General > General preference settings" on page 863, Chapter , "Configuring iTest Preferences") or select a different language from the list.

4. Specify the **Session type** (**Telnet**, **SNMP**, **Web**, and so on). This is a default iTest template for a session. If you will use the session type as the basis for this session configuration, then skip the next step and continue at Step 6.

5. In this step, you specify that the current session configuration should inherit all property settings from an existing session profile — either a reference session profile or any other session profile. This is the most common and the most powerful way to define a session profile, as described in "Defining a reference session profile" on page 77.

   a. Check **Use a reference session profile.**

   b. In the **Create reference session profile** box, specify the session profile from which to inherit settings.

      When you click **Browse**, the dialog box enables you to select from both reference and non-reference session profiles, either in the **Workspace** (the document is in the current iTest workspace) or **File system** (the document is in an itar file somewhere in the file system).

6. The current property settings appear in the middle of the page. Modify the settings as needed. For an SNMP session, for example, you would specify the **IP address** for the device that you're connecting to. Other session types have different required settings. The property settings associated with each session type are described in the chapter for the particular session type in a section titled "Session profile property settings for <session type> sessions".

- Required property settings are marked with the * character.
- A blue text box indicates that the setting is being inherited from the session profile that this session profile is based upon. See "About property settings" on page 72.
- The F indicates that you can use field replacements in the text of the property setting. See "Field replacements: Substituting values into properties and commands" on page 631.

You have the option to specify additional session properties like screen color, timeouts, non-standard prompts, and so on. Click More >> to display a tree of all available properties. you will find details on property settings for each session type in the appropriate chapter.

7 Click **Save** to save the new session configuration with the topology document. Click **Save and Start** to save the definition and start a session with the device.

To update a session configuration for a resource

On the canvas, right-click the resource and select **Edit Session**.

(Alternatively, select the device on the canvas. On the Properties view, click the **Session** tab and then click)

The **Edit Session Profile** page enables you to view or edit property settings for the session profile.

To remove a session configuration for a resource

On the canvas, right-click the resource and select **Remove Session**.

(Alternatively, select the device on the canvas. On the Properties view, click the **Session** tab and then click)

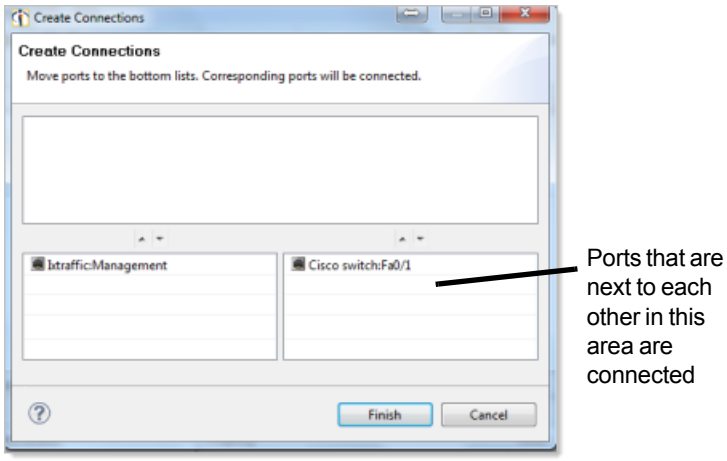
To quickly add multiple links (connections between ports)

Note The devices that you work with must each have at least two ports.

1 Right-click a device and select **Create Connections**.

2 In the list at the top, open a device and then drag-and-drop one or more ports onto the connection area at the bottom.

3 When you have created all connections, click **Finish**.



TBML Command Wizard

The TBML Comamnd wizard takes you through the creation of all of the tbml commands, for example:

- Find a resource property value: The wizard will show a table of resources and their properties for you to select.
- Find endpoints of a link: The wizard will display a list links to select.

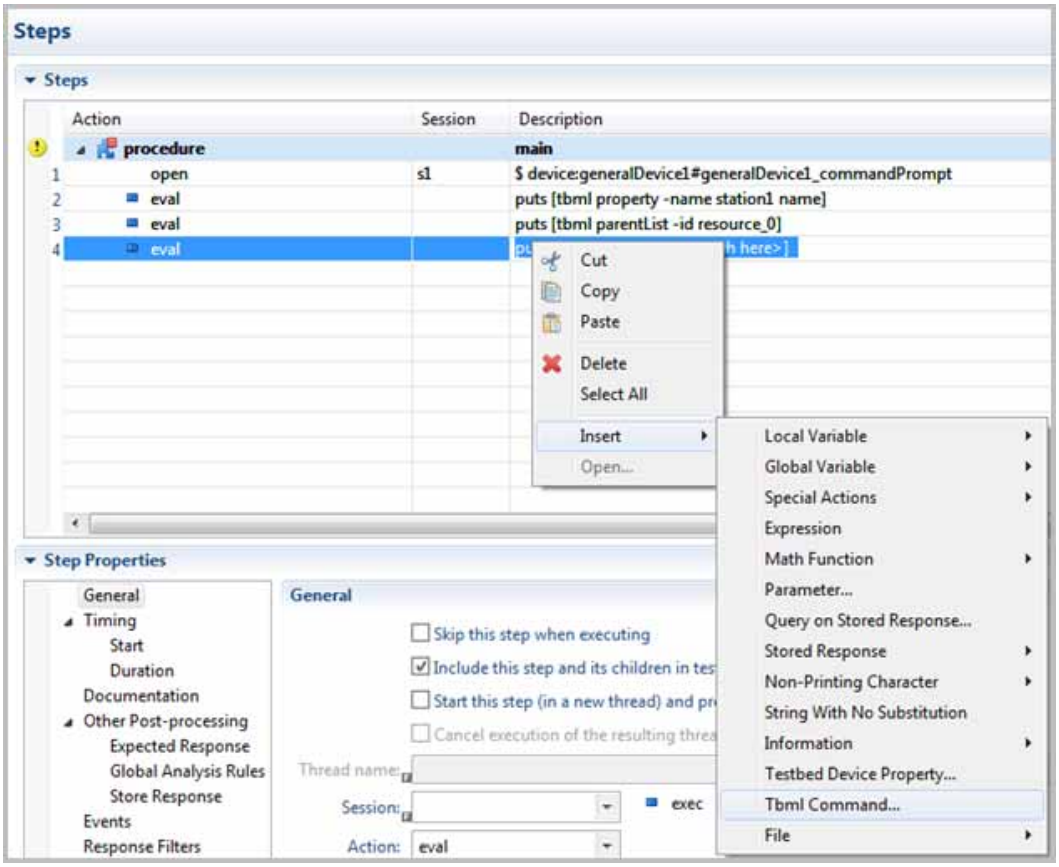
After making your selection, the wizards builds the command automatically. You are not required to know or understand understand the TBML command syntax.

Using the TBML comamnd wizard

After defining the topology, open the **General** page on the Test Case editor and specify the **Local topology** for the test case.

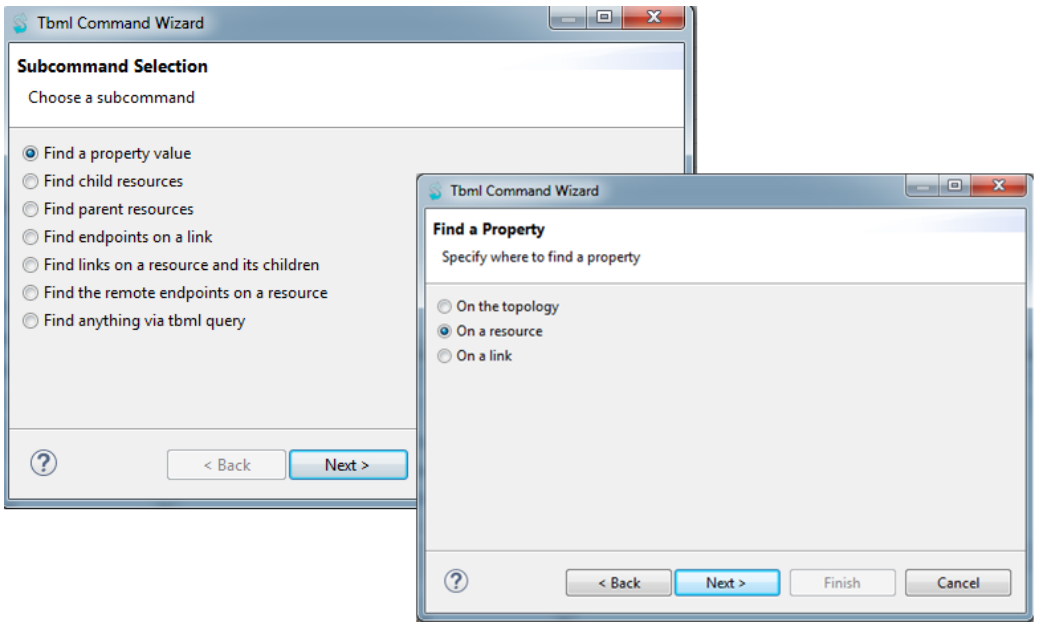
- For **open** steps in the test case, select any of the devices in the topology from a drop-down list.

- Add a Step to access the Parameters defined in topology session profiles, for example on a new Step, right-click, select **Insert > TBML Command** option.



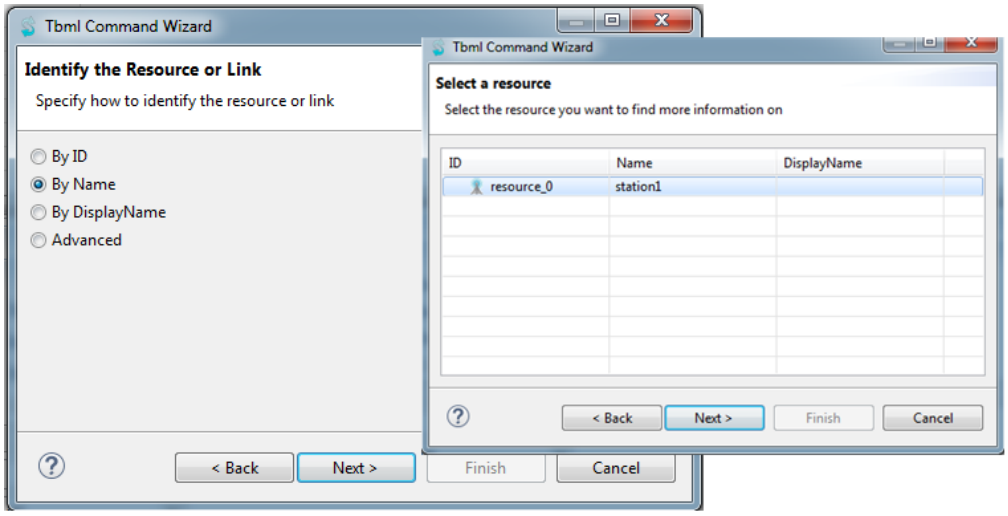
Note The following steps are included as an example and the options displayed on the **TBML Command Wizard** dialogs may vary depending on your selection.

- The **TBML Commands Wizard** opens. Select a **Subcommand** and click **Next** and select a property on the **Find a Property** dialog..



Note Selecting the option **Find anything via TBML query** displays the TBML command on the **Finish** dialog, which may be inserted in the Test Case Step.

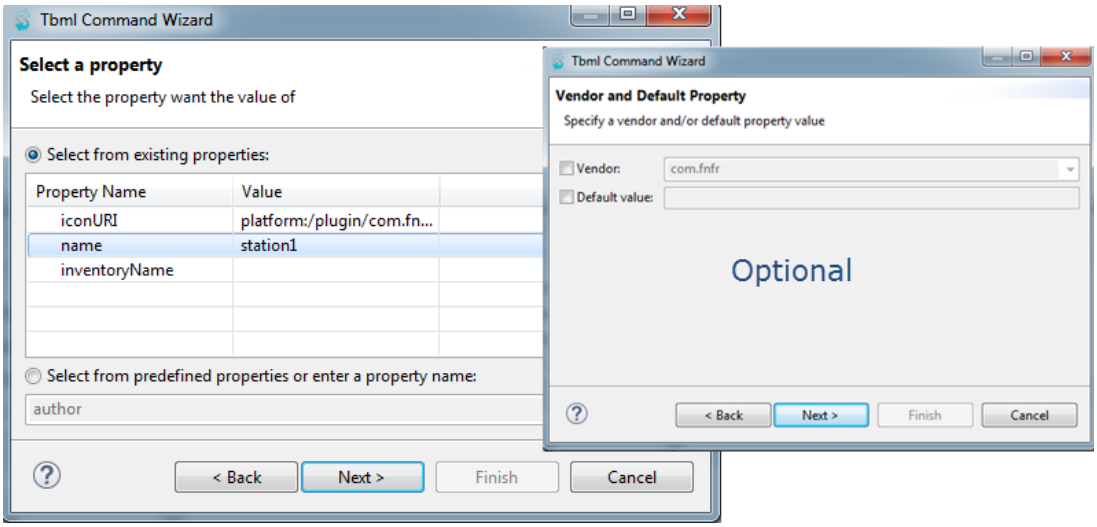
- Click **Next** to Open the **Identify the Resource or Link** dialog and click **Next** and select the required resource or link.



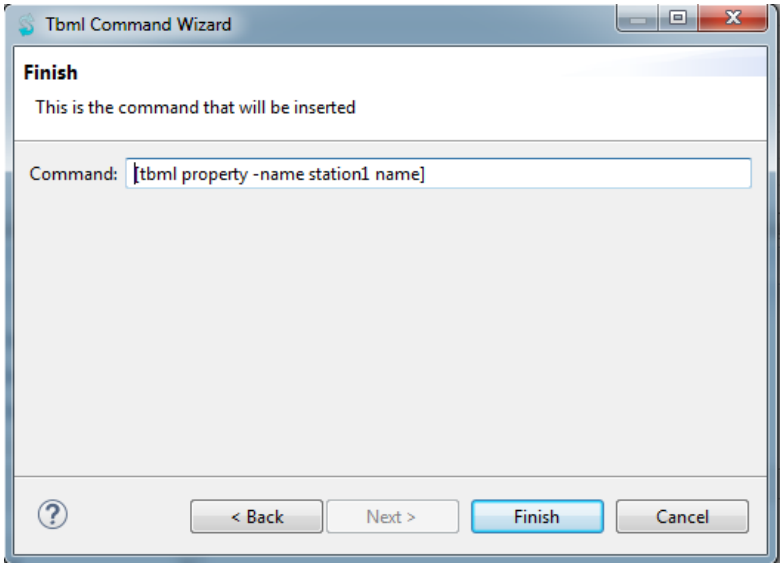
Note The **Next** button appears only when you select a resource or link.

Click **Next** on the **Select a resource** dialog.

- **Select a property** on the dialog or select from a predefined properties or enter a property name. Click **Next,** define the **Vendor** and **Default Property** value if required, and click **Next** again.



- The **TBML Command Wizard** displays the command line generated based on your selection on the **Finish** dialog. Click **Finish** to insert the dialog into the Test Case Step.



You may execute this Test Case as all other test and see the result.

Migrate Testbed to Topology

iTest also provides a conversion wizard to easily convert your existing Testbed to Topology, which you may edit as required. You may convert existing Testbeds in these two ways.

- **On the Testbed editor: Devices page** (which is made up of devices), right-click, and choose the **Migrate to Topology** action.,

Migrate to Topology	Click **Mirgrate to Topology** to mirgrate the testbed to a corresponding topology. For example, test_testbed.tbml.

- Go to i**Test > tools > Convert Testbed to Topology**

iTest creates a corresponding topology and opens in the topology editor. Follow the instructions in "Topologies: Quick instructions" on page 493 to edit the topology as required.

Working with Velocity

Connect to Spirent iTest server

1. On the **Preferences** page (**Window > Preferences**), navigate to **Spirent > Velocty.**
2. Specify the following settings:

Server URL	Specify the URL or hostname of the host where Velocity is running.
Username / Password	Optional: Specify the default username and password to use to log in when iTest starts.
Sync Interval (sec)	Velocity periodically checks for changes on the Velocity server (topologies, resources, and reservations) to ensure that the data is always "in sync" with iTest. Specify the time interval between data refreshes. Default: 30 seconds
Login Mode	Select to indicate the login mode to Velocity when iTest starts: Automatically Login: Prompt for every login Do not auto-login at start up

3. In the palette, under **Velocity**, click **Not connected**. In the **Login** dialog box, specify the Spirent iTest server and your login credentials.

Publish a topology

When you publish a topology that was created in iTest and upload it to Velocity, all of the resources defined in the topology become available for reservations in Velocity.

1. If iTest is not connected to Velocity: In the palette, under **Velocity**, click **Not connected**. In the **Login to Velocity** dialog box, specify the Velocity location and your login credentials.

(If the Spirent iTest settings have already been configured on the **Preferences** page, then you need to provide only the password).

When the connection is established, the palette displays the Spirent iTest inventory of devices arranged in folders. If this is the first topology that you have published, then the folders might be empty.

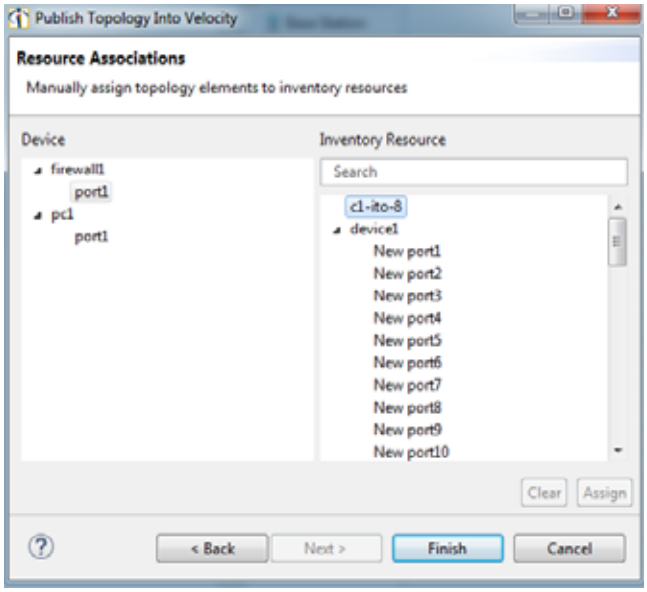
2 Right-click the topology in the iTest Explorer and select **Velocity > Publish**.

3 Specify the following settings:

Name	Specify the name to use when listing this topology. If you are publishing an abstract topology, then type a meaningful name.
Physical Topology	This line identifies the physical topology that is the basis of the topology that you are publishing. iTest supports one physical topology.
Manually assign resource associations	Check the box to open another wizard page to assign particular resources for the resources specified in the topology that you are publishing.

- If you are publishing an abstract topology, then click **Finish** and skip to Step 5 on page 507.
- If you are publishing the physical topology, click **Next**.

4 If you are publishing the physical topology, then the **Resource Associations** page opens to enable you to specify how the devices in the TBML file map to the resources defined in Spirent iTest.



5 Once the topology is published to Velocity, Spirent iTest tags the topology as *synchronized*, meaning that, before the topology can be used in iTest, Spirent iTest first verifies the reservation status of all devices in the topology.

Update a published topology

Spirent iTest uses a source control mechanism to ensure that topologies are synchronized between iTest and Velocity so that Velocity always uses the most up-to-date version of a

topology. You must ***lock*** any published topology to enable editing and, when the changes are complete, must ***commit*** the updated topology to Velocity. Velocity then ***unlocks*** the topology again.

1 With the topology open in the Topology editor and while connected to **Velocity**, right-click anywhere on the canvas and select **Lock**. Now, only you can modify the topology.

2 Make changes to the topology as needed.

CAUTION When you delete a resource or port from the topology, then all logical topologies that used to contain the resource become invalid (because the topologies refer to a resource that no longer exists). See the Velocity help for detailed information on logical topologies.

3 Right-click anywhere on the canvas and select **Commit**.

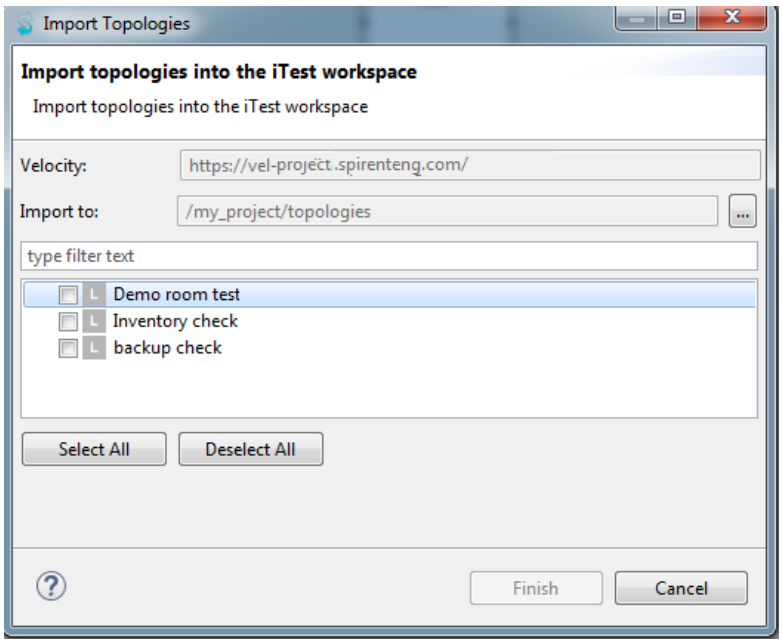
Click the **Undo changes and unlock** button to discard any changes you made.

Import a topology from Velocity into your iTest workspace

When a topology is defined in **Velocity**, you must import it into the iTest workspace to use the topology in test cases.

When you import a topology, iTest creates a copy of the topology (**.tbml** file) at the specified location. If any changes are made to resources in the topology, the system auto-syncs the data between iTest and **Velocity**.

1 While connected to **Velocity**, in the iTest Explorer, right-click the folder in the workspace where the topology should be saved (typically the **Topologies** folder). Select **Velocity > Import Topologies.**



2 In the **Import Topologies** dialog box, select the topologies to import.

The Search box enables you to quickly find specified items. The search string and names you enter are converted to same case (lower case) and search begins from first letter of the name of the item or any word within the full name. For example, :

Search string = check, will match and display as follows:

Inventory check

backup check

Search string: test, will display as follows:

Demo room test

3 Verify the location to import to, and then click **Finish**. You can import only logical topologies.

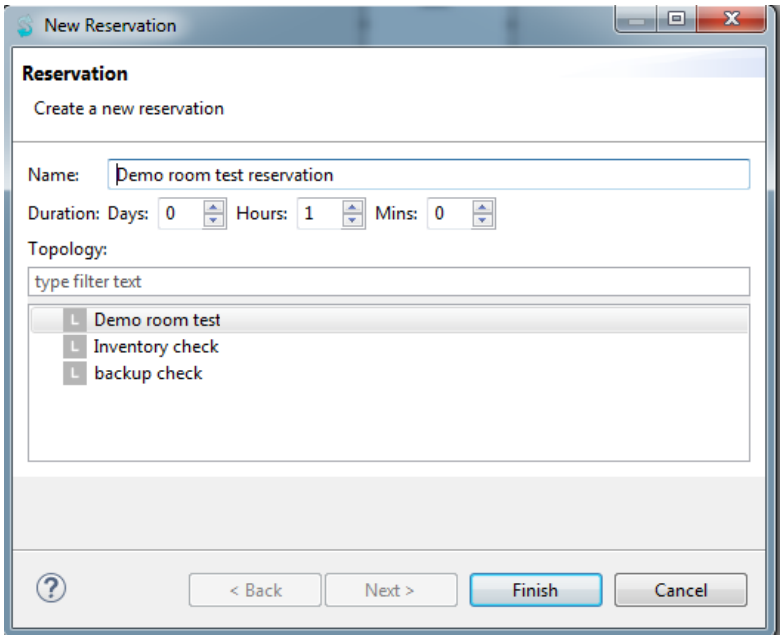
Tip Use **Select All** / **Deselect All** to select a group of topologies to import at one time.

Reserve a topology

In iTest, you can create a reservation that starts immediately. To schedule a reservation for a time in the future, use **Velocity**.

1 With the topology open in the Topology editor and while connected to a **Velocity** server, right-click anywhere on the canvas and select **Reservations > Reserve**.

2 On the **Reserve** dialog box, specify the duration and then click **Finish**.



The Search box enables you to quickly find specified items. The search string and names you enter are converted to same case (lower case) and search begins from first letter of the name of the item or any word within the full name. For example, :

Search string = check, will match and display as follows:

Inventory check

backup check

Search string: test, will display as follows:

Demo room test

You might sometimes request a resource that is not available at the requested time; it might already be reserved for some or all of the requested duration. In this situation, iTest notifies you by displaying the **Scheduling Conflict** dialog box.

Release (cancel) a reservation

To release (clear) all port mappings that are being used by a reservation (for example, if you no longer need to use the resources), right-click in the palette and select **Reservations > Release**.

Note Port mappings that are currently being used by another active reservation are ***not*** cleared.

View reservations

- **To filter the list of reservations**

  Use the filter option to narrow the list of reservations that appear in the list based on the **Start time** and/or the **End time** for the reservation:

  1 On the Spirent iTest **Reservations** view, click **Filter** .

  2 Reservations appear on the Spirent iTest **Reservations** view based on the settings that you specify:

Specify both Start time and End time	The following reservations are listed: The starting time for the listed reservation is between (or the same as) the specified **Start time** and **End time** or The ending time for the listed reservation is between (or the same as) the specified **Start time** and **End time**
Specify only the Start time	The following reservations are listed: The ending time for the listed reservation is the same as or later than the specified **Start time**
Specify only the End time	The following reservations are listed: The starting time for the listed reservation is the same as or earlier than the specified **End time**
Show inactive reservations	Display reservations that have completed or have not yet started.

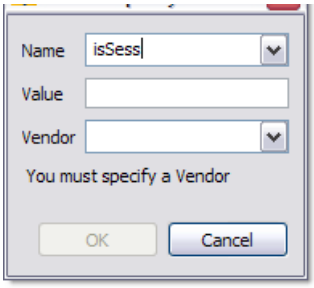
General Topology Operations

Specify that a device requires or does not require a session profile to be defined for it

By default, devices that you add to a topology require that you define a session profile (that is, the **isSessionRequired** property is **true**). By default, the **isSessionRequired** property does not appear in the in the list on the **Device** tab. If you change its value, then the property appears in the list.

Note The **isSessionRequired** property setting is used only if the **Vendor** property for the topology is **com.fnfr**

If you explicitly add the **isSessionRequired** property, the property does not appear in the list for the **Name** setting. You must type the text "**isSessionRequired**" and select the **Vendor** property value of **com.fnfr.**



- **To specify that a device requires or does not require a session profile**

  Right-click the device on the canvas

  - If **Device Requires Session Profile** is checked, then the **isSessionRequired** property is **true.**
  - If **Device Requires Session Profile** is not checked, then the **isSessionRequired** property is **false.**

Control emulation for devices in a topology

You can control how emulation is used for any device in a topology in any test case that uses the device. For details, see "Controlling emulation for devices in a topology" on page 484.

Topology editor: Properties view, Topology tab

Topology properties that communicate design history

Use the **Topology** tab to document the topology's design history for the benefit of coworkers. Most values are auto-generated. You can specify the following values:

creationTool	
creationToolVersion	
dateModified	
@Description	Optional. Type text that describes the topology to make its usage clear to coworkers. The @ symbol denotes that this value is a TBML attribute.
@Name	Optional. Type a one-line "friendly" name for the topology. For example, **LabA_Switch_environment** or **Routers,TrafficGen, and Linux box** This text appears in the Favorites view for the topology. The @ symbol denotes that this value is a TBML attribute.

Topology tab toolbar

	Add a new property or property collection to the topology. See "Add a property to a topology, to a device, or to a link" on page 496 and "Add a property collection to a topology, to a device, or to a link" on page 497.
	Delete the selected property.

Topology editor: Properties view, Session tab

Select any device to cause the Session tab to appear. Use the **Session tab** to add, delete, view, and edit device sessions. The properties are set when you add a session profile as described in "Add, edit, or remove a session configuration for a topology device" on page 500.

Session tab toolbar

	Add a new session profile for the selected topology element. The **Add Session Profile** page takes you through the process. See "Add, edit, or remove a session configuration for a topology device" on page 500.
	Delete the selected session profile definition. (Use Ctrl-click or Shift-click to select multiple session profiles.)
	Start the selected session in a new window. . (Use Ctrl-click or Shift-click to select multiple session profiles.)
	Open the **Edit Session Profile** page to view or edit property settings for the session profile. (Use Ctrl-click or Shift-click to select multiple session profiles.)
	Common editing tools. **Cut**, **Copy**, **Paste** the selected device sessions within the current topology or to another topology. (Use Ctrl-click or Shift-click to select multiple session profiles.)

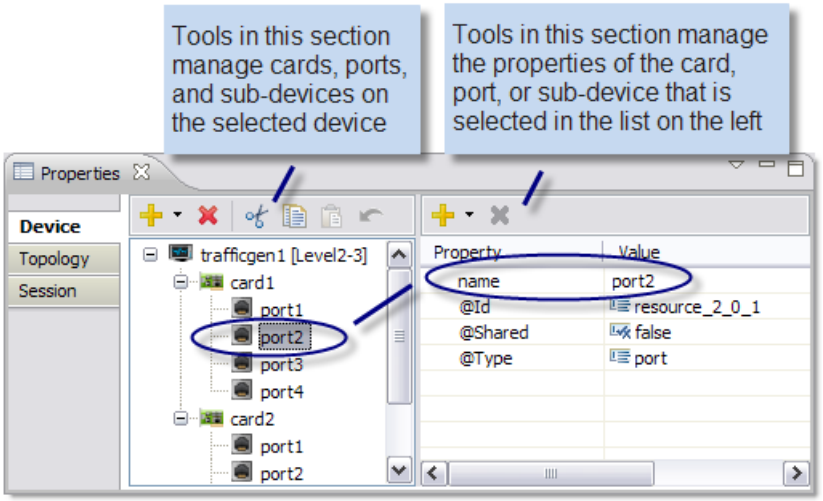
Note When you use the Topology editor to define a session profile for a device, you cannot specify a value for the session profile **Session name** property.

Name	Type the name for this particular set of session configuration settings. The name is used as the default session name when you create a test case from captured steps or add an open step to a test case. The name appears in the Session column for each step in the session
Type	The session type; Telnet, SSH, Web, SNMP, Spirent TestCenter, Serial, and so on.
Base session	The session profile that this device session inherits its property settings from
IP Address	The **IP address** or **hostname** for the device
Port	the port used to communicate with the device or software
User ID	Some session types require user authentication to start a session If required, you specified the **User ID** while configuring the device session profile.

Password	Some session types require user authentication to start a session If required, you specified the **Password** while configuring the device session profile. The text is masked here and in all locations in which it is used
URL	Web-based session types offer the option to specify the URL to use when starting a session You specify the **URL** while configuring the device session profile. The text is masked here and in all locations in which it is used. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

Topology editor: Properties view, Device tab

Select any device to cause the **Device** tab to appear. Use the **Device** tab to add cards or sub-devices to the device and to add ports to the device, to a sub-device, or to a card. In addition, you can specify property and attribute values for devices, cards, and ports. .



Note Generic Devices (the kind of sub-device that you can add to a device) have no default **@Type** attribute value. You can set the attribute value as needed (for example, **workstation**),

Device tab toolbar (Card, Sub-device, and Port section)

	Add a Card, a Port, or a Generic Device to the selected device. iTest provides a unique default name. The properties of the new item appear in the section to the right. See "Add a card, a port, or another device to a device" on page 495.
	Delete the selected items. (Use Ctrl-click or Shift-click to select multiple sessions.)

	Common editing tools. **Cut**, **Copy**, **Paste** the selected items within the current topology or to another topology. (Use Ctrl-click or Shift-click to select multiple sessions.)
	Convert the resource to an abstract resource

Device tab toolbar (property section)

	Add a new property or property collection to the selected item. See "Add a property to a topology, to a device, or to a link" on page 496 and "Add a property collection to a topology, to a device, or to a link" on page 497.
	Delete the selected property and its value.

Device page context (right-click) menu

Copy	Common editing tool. **Copy** the selected item for pasting within the current topology or in another topology.
Restore Default Value	Typically, you do not need to alter this setting. Restores the selected property to its original TBML schema value.

View resolved resources properties (from Abstract Topology)

You may view the resolved resources from Abstract topology in iTest GUI. That is, view the list of concrete resources allocated for use during reservation in the specified time range.

To view the resources allocated to (Abstract Topology) during a reservation:

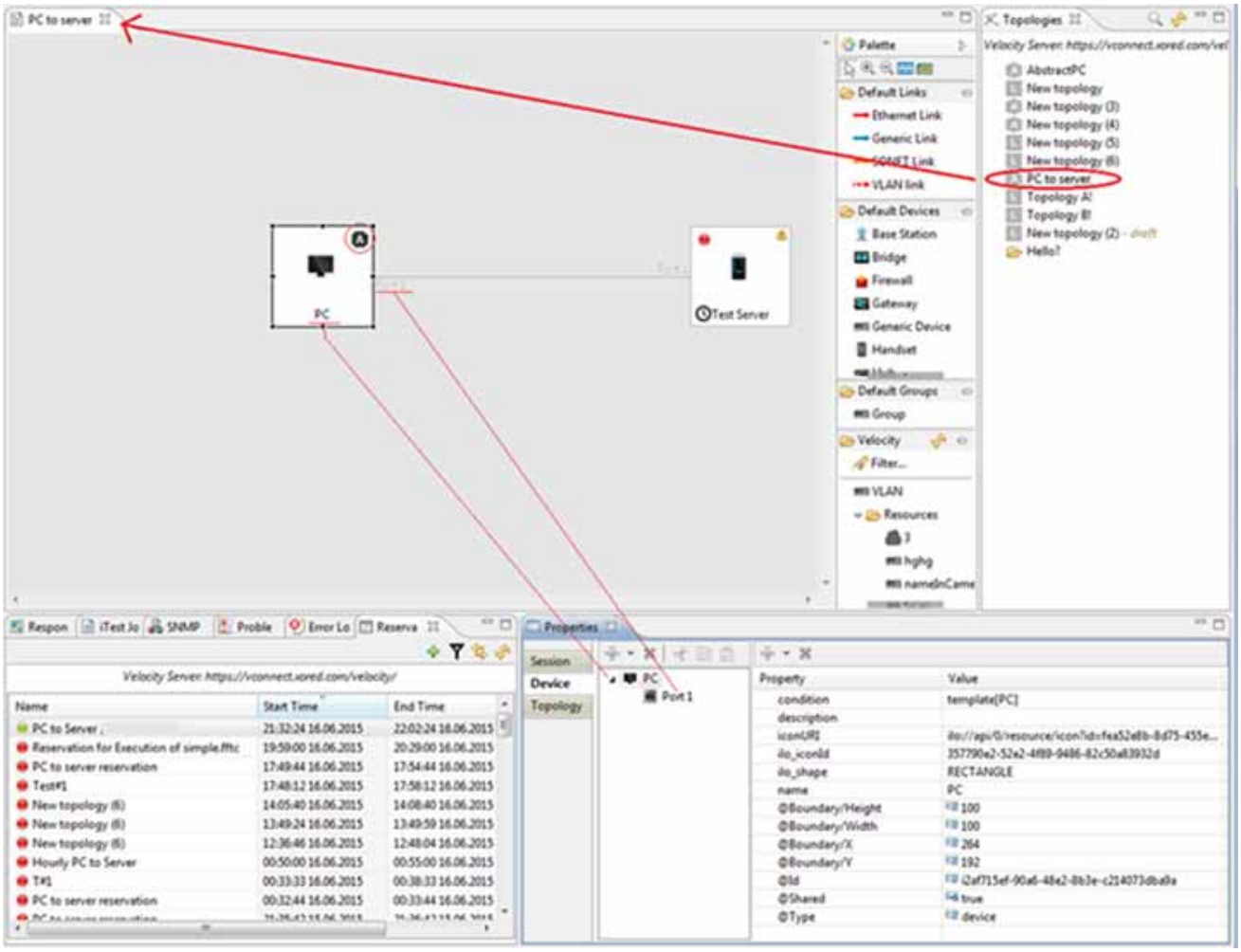
- View the list of reservations displayed.

Note You may set up a filter to list abstract topology with both active and inactive reservations.

- From Reservations view, select the required Abstract Topology and open a reserved topology (double clicking on the topology or use the context menu).

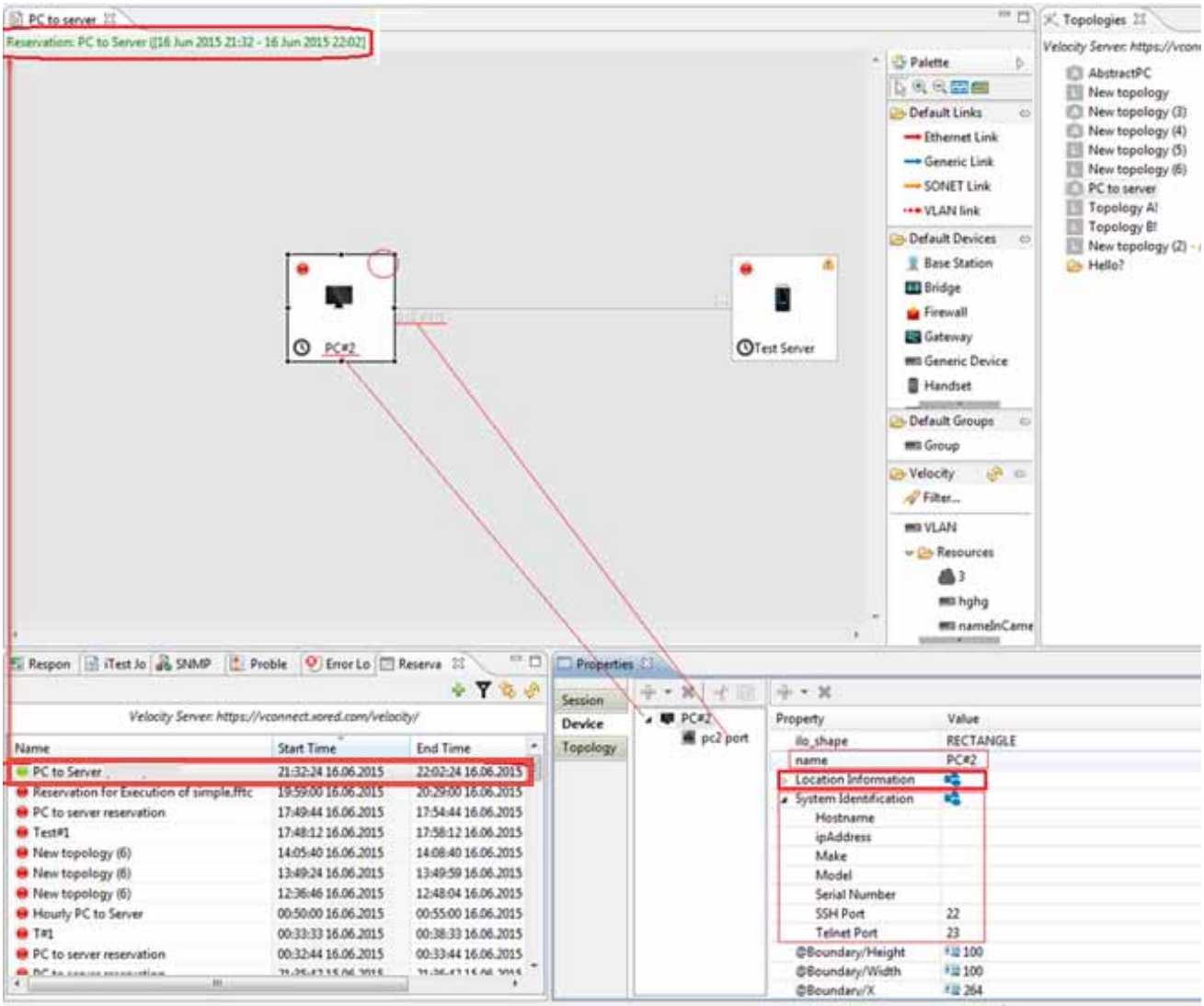
Note Reserved topology displays reservation name, start time and end time.

- The Reserved topology opens. The example below illustrates an Abstract Topology. In Topologies view, double-click and open to view.



In the above illustration, **A** on the device named PC indicates an abstract device. The Properties view does not contain some properties, for example, **System Informations**, which is available only once the reservation is active and resources are allocated.

- Double click the topology (from the Reservations view) to open it with resolved resources for the current reservation. The diagram below illustrates the topology with resolved resources.
  - In the properties view, the resolved value/system identification details display when device is selected.
  - Resolved port details display when port is selected.



Note

- The names of resolved resources show in the above diagram. Properties view contains additional properties assigned with concrete allocated resources.
- The topology can have several reservations. Each reservation will open in a separate window (topology editor).
- View the resolved resources for:
  - Active reservations.
  - Completed reservations (past reservation).
  - Scheduled reservations (future reservation)

For recurring reservations, resources will be allocated at the start time, i.e., it is not possible to view them in advance. Double clicking an inactive recurring reservation will open an abstract topology.

Topology editor: properties, tbml

A reserved abstract topology includes the tbml properties for device identification.

Note See also "View resolved resources properties (from Abstract Topology)" on page 515.

iconURI	Refers to the URI of the device icon
name	Refers to the abstract or logical name (e.g, router1)
logicalName	Refers to the physical resource name.
inventoryName	Applicable to Velocity resources. • For **Concrete Velocity device**: The **inventoryName** refers to the phical resource name • For **Abstract Velocity device**: The **inventoryName** will be added to device/resource when the topology is reserved and resources allocated (resolved).
resolvedName	Refers to the physical name.(resource) allocated to an abstract resource when the reservation starts.
Location	Refers to the physical location of the resource
System Identification	Refers to the hostname, IP address, make, model, and serial number of the resource

Topology editor: Properties view, Link tab

Select any link (connection between ports) to cause the **Link** tab to appear. Use the **Link** tab to add properties or property collections and to set property and attribute values for the link.

Also, see "Add and configure a link between two devices" on page 495 and "Add a link from a device to itself" on page 495.

- **Setting the Source and Target devices for a link**

  Click in the **Value** cell for the **Source** or **Target** property and then select the appropriate port or card.

- **Managing properties for the link**

	Add a new property or property collection to the selected link. See "Add a property to a topology, to a device, or to a link" on page 496 and "Add a property collection to a topology, to a device, or to a link" on page 497.
	Delete the selected property and its value. **Note** The **Source** and **Target** properties cannot be deleted.

Topology editor: Properties view, Ruler and Grid tab

Use the **Ruler and Grid** tab to specify how new devices should align when added to the canvas, whether to display the ruler, and to specify the appearance and operation of the alignment grid.

'Velocity' command

Commands that return information from Velocity

The **Velocity** group of commands returns information about resources and ports on the topology that is active during the current reservation.

See "About Tcl commands" on page 342 and "About Python commands" on page 355 in Chapter 17, "iTest Commands".

Important:

Using square brackets ([]) in Python syntax:

- Required in session steps and session profiles fields.
- Not required in non-session steps (eg: eval).

Note Square brackets are required if using Python and Velocity subcommands, when editing a session profile of a device in a topology imported from Velocity.

velocity command syntax

Tcl: velocity *subcommand arg ?arg?*

Python: velocity*(velocity('subcommand', 'arg') #*args*

Specifying arguments

- In the command descriptions, "resource" means a device or any item in a nested device/sub-devices/cards/ports structure.
- For many commands, to make it easy to specify a resource or link, you can specify the it by its **id** or by its **name**
- When specifying a resource path as an argument, the parent is listed fist, followed by the child. For example, **"myRouter card1"**

property subcommand: Return value of a property

The **velocity property** command returns the value of a property on a resource or on a link.

- To identify a property, use its **id** or **name**

Issues

- If a resource or link with the specified **id** or **name** is not found, and a default value is not specified using the **-default** (**default** in Python) argument, then iTest generates an **onInterpreterError** execution issue.
- In the event that the resource was not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.

- The specified resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

velocity property -id *ID propertyName* **velocity('property', '-id**', *'ID', 'propertyName'*)	For a resource or link with ID of *ID*, returns the value of the property with *propertyName.* **Example** **velocity property -id resource_0 name** **velocity('property', '-id', 'resource_0', 'name')** returns the value of the **name** property for the resource with ID **resource_0**.
velocity property -name *name propertyName* **velocity('property', '-name**', *'name', 'propertyName')*	Returns the value of the property specified by *propertyName* for the resource or link specified by *name.* **Example** **velocity property -name myRouter myCustomProperty** **velocity('property', '-name', 'myRouter', 'myCustomProperty')** 1. First, finds the resource **card1**.We specified the full name path starting from the top level resource which is **myRouter**. 2. Then finds the value for **myCustomProperty**, which is a child of the property collection group called **my property collection**.

allPortList subcommand:

The **velocity allPortList** subcommand returns the concrete ports that are mapped for the device during a reservation.

To make the return value useful as a session property that requires port list, the return value is a string value of comma-separated port numbers.

Example Tcl: **[velocity allPortList -id resource_0]**

Example Python: **velocity('allPortList', '-id', 'resource_0')**

You can specify the device in the same way as for the **property** subcommand.

makeReservation subcommand

The velocity **makeReservation** subcommand returns the reservation and topology ID (reservation outside the scope of iTest).

Tcl: **velocity makeReservation -topologyName <topologyName> -duration <duration> [-name <reservationName>] [-- <conditions>]**

Python: **velocity('makeReservation', '-topologyName', '<topologyName>', '-duration', '<duration>', '-name', '<reservationName>')**

Arguments:

topologyName	Name of the topology to reserve (string, required)
duration	Reservation duration in minutes (number, required)
Name	Name of the reservation (string, optional)

Conditions (listed them separate from arguments):

Conditions	List of objects representing custom conditions for abstract resources. (list, optional)

Important

- Each conditions must be within double quotes ("---")
- Each conditions is paired with resource name and condition value: **<resource_name>.cond=<condition_value>**
- The next symbols must be represented using the back slash:

  '[', ']', '"' '\'

Note See **Velocity Abstract Resource Conditions Language Reference Guide** for additional information about condition string.

Result:

Result	Reservartion id **Note** You may user "**store response**" settings to store this reservation id..

Example Tcl usage:

velocity makeReservation -topologyName "YK1" -duration 10 -- "PC.cond=template\[PC\] and ports(integer\[Port Speed\]>=10000)>=2" "Server.cond=template\[Server\] and \[Hostname\]=\"xxxxx.xxxxx.com\""

Example Python usage:

velocity('makeReservation', '-topologyName', 'YK1', '-duration' '10', 'PC.cond=template\[PC\] and ports(integer\[Port Speed\]>=10000)>=2', 'Server.cond=template\[Server\] and \[Hostname\]=\"xxxxx.xxxxx.com\"')

cancelReservation subcommand

The velocity **cancelReservation** subcommand returns identifier of the reservation stopped/canceled.

Tcl: **velocity cancelReservation -reservationId <reservationId>**

Python: **velocity('cancelReservation', '-reservationId', '<reservationId>')**

Arguments:

reservationId	Identification of reservation: (UUID, required) For example, UUID is the result of the **makeReservation** command

Example usage Tcl:

velocity cancelReservation -reservationId $reservationId

Example usage Python:

velocity('cancelReservation', '-reservationId', 'reservationId')

findReservations subcommand

The **findReservation** command returns reservation list by filter.

Note The Velocity server limits the list of reservations returned to 200.

Command Tcl

velocity findReservations [-name <reservation_name>] [-status <status>] [-owner <owner>] [-startTime <startTime>] [-endTime <endTime>]

Command Python

velocity('findReservations', '-name', '<reservation_name>', '-status', '<status>', '-owner', '<owner>', '-startTime', '<startTime>', '-endTime', '<endTime>')

Arguments:

name	Name of the reservation. Search with using wildcard. I.e. use substring in place of real reservation name. (String, optional)
status	The following lists the valid values (string, optional) : • **STANDBY**: Reservation will wait until the work order is completed. • **PENDING_APPROVAL**: Valid only for future reservations, that is, the reservation is waiting for approval and will fail if not approved. • **SCHEDULED**: Reservation is scheduled for a future date and time. • **ACTIVATING**: Reservation is in the process of being activated. That is, the activation procedures are in progress (including startup tasks). • **ACTIVE**: Reservation is currently active. • **DEACTIVATING**: The deactivation procedures are in progress (including teardown tasks). • **COMPLETED**: Reservation was completed (past). • **SUSPENDED:** Applies only for recurring reservations; For example, Canceling a recurring reservation cancels future events, and the reservation shows as in a **Suspended** status. • **DECLINED**: Reservation was declined • **FAILED**: Reservation failed due to an of these reasons: canceled by a user, not approved before start, and so on. • **UNKNOWN**: Applies to private reservations. That is, the details are hidden form the requesting users.
owner	Creator of the reservation (string, optional)
startTime	Search reservation that start after specified time. time format: HH:mm:ss dd.MM.yyyy Example: 08:05:00 21.09.2015 t<number> - offset from the current time in minutes Examples: 10 minutes later: t10, 10 minutes ago: t-10
endTime	Search reservation that end before the specified time
Result (List of pair)	: **<reservation_id>**SPACE**<reservation_name>**

Example usage:

Tcl: velocity findReservations -status ACTIVE -name YK1 -endTime t60

Python: velocity('findReservations', '-status', 'ACTIVE' '-name', 'YK1', '-endTime', 't60')

setReservation

The **setReservation** subcommand allows you to specify (programmatically) an active reservation ID to be used in a test case so that you can run a test case against a reservation made outside the scope of iTest.

For example, assign reservation with current testcase:

Tcl: velocity setReservation -reservationId <reservationId>

Python: velocity('setReservation', '-reservationId', '<reservationId>')

Arguments:

reservationId	Identification number of the reservation. For example, it can be the result of "findReservations subcommand" on page 522. (UUID, required) **Note** The **findReservation** command returns a list of **pair<id, name>**, but only the id is required. For example, in order to list the **id** of the first reservation in list, use: **eval "lrange [split $reservation] 0 0".** In Python use:

Example usage:

Tcl: velocity setReservation -reservationId $reservationId

Python: velocity('setReservation', '-reservationId', $reservationId)

reservedPortList subcommand:

Example Tcl: **[velocity reservedPortList]**

Example Python: **velocity('reservedPortList')**

The **reservedPortList** subcommand retrieves all topology ports (which might be concrete or abstract) of the specified device and then returns the list of their mappings. The return value format is the same as for the **allPortList** subcommand.

You can specify the device in the same way as the **property** subcommand.

reservationId subcommand

Example Tcl: **[velocity reservationId]**

Example Python: **velocity('reservationId')**

The **velocity reservationId** subcommand retrieves information about the specified reservation.

token subcommand

Example Tcl: **[velocity token]**

Example Python: **velocity('token')**

The **velocity token** subcommand retrieves authentication token that may be used for authentication in **Velocity REST API** (returned token expires after 24 hours).

url subcommand

Example Tcl: **[velocity url]** subcommand

Example Tcl: **velocity('url')** subcommand

The **velocity url** returns URL of the associated Velocity instance.

Note You may use [velocity] subcommands/velocity(subcommands) when authoring Velocity drivers.

‘tbml’ topology commands

Commands that return information about topologies

The **tbml** group of commands returns information about resources from a topology (that is, information about a device, a link, or the topology itself).

tbml command syntax

Tcl: **tbml** *subcommand arg ?arg?*

Python: **tbml***('subcommand', 'arg') #*args*

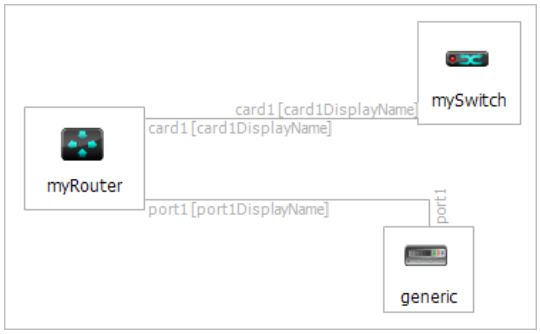
Specifying arguments

- In the command descriptions, “resource” means a device or any item in a nested **device/subdevices/cards/ports** structure.
- For many commands, to make it easy to specify a resource or link, you can specify it by its **id**, by its **name**, or by its **displayName**.
- When specifying a resource path as an argument, the parent is listed first, followed by the child. For example, **“myRouter card1”**
- Segments in a resource path are delimited (that is, a resource is separated from its child) using the space character. If there is more than one segment in the path, then you must surround the path with double quotes. For example, to refer to **port1** on **card1** on **myRouter**, use **“myRouter card1 port1”**
- If a resource name includes a space character, then pass the arguments as a list. For example,

Tcl example	Python example
Pass a resource named **blue Router** as **[list “blue Router”]**	Pass a resource named **blue Router** as **resource_name = 'blue Router'**
Pass a resource named **card1** on **blue Router** as **[list “blue Router” card1]**	Pass a resource named **card1** on **blue Router** as **resource_name + = 'card1'**

Example topology

In the subcommand descriptions, we refer to the following topology:



Resource	Property		Value
Topology	description		
	my collection	myProperty	
		……	
myRouter	name		myRouter
	displayName		myRouter_displayName
	Id		resource_0
	author		Vidya (default)
myRouter/card1	name		card1
	displayName		card1_displayName
	Id		resource_0_0
	my property collection	myCustomProperty	hello world
		……	
myRouter/card1/port1	name		port1
	displayName		port1_displayName
	Id		resource_0_0_0
	ipAddressV4		10.100.2.3
mySwitch	name		mySwitch
	displayName		mySwitch_displayName
	Id		resource_1
mySwitch/card1	name		card1
	displayName		card1_displayName
	Id		resource_1_0

property subcommand: Return value of a property

The **tbml property** command returns the value of a property on a resource or on a link.

- The **property** subcommand can return multiple values if multiple properties match.
- Because the **property** subcommand applies to both resources and link elements, the resource element takes precedence over link elements. If a property is not found in a resource element, then the link element is searched.
- To identify a property, use its **id** or **name** or **displayName** or a combination of **id** and one of **name** or **displayName**.

See "tbml command syntax" on page 524

Note When using **tbml** commands, any strings that include spaces need to be surrounded with curly braces.

The following examples show the correct and incorrect usage:

Correct syntax	`tbml property -id $deviceId {{System Identification} ipAddress}`
	`OR the following in Python`
	`tbml('property', '-id' , 'deviceId', 'ipAddress')`

Incorrect syntax	`tbml property -id $deviceId {"System Identification" ipAddress}`
	`OR`
	`[tbml property -id $deviceId "System Identification" ipAddress]`
	`or`
	`tbml('property', '-id' , 'deviceId', ipAddress)`
	Error:
	`Unable to substitute " Device IP: tbml property -id $deviceId {"System Identification" ipAddress}": Cannot find property: com.fnfr.svt.iTestInterpreter.model.SubstString@e19679/ipAddress`

Issues

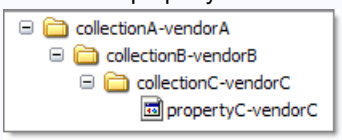
- If a resource or link with the specified **id**, **name**, or **displayName** is not found, and a default value is not specified using the **-default** (Tcl) **default** (Python) argument, then iTest generates an **onInterpreterError** execution issue.
- In the event that the resource was not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.

- The starting resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

tbml property	Returns property value, Id, etc., as specified
tbml property -id *ID propertyPath* **tbml('property', '-id'**, *'ID', 'propertyPath')*	For a resource or link with ID of *ID*, returns the value of the property at *propertyPath.* *propertyPath* is the path to a property using the name of the property. **Example** **tbml property -id resource_0 name** **tbml('property', '-id', 'resource_0', 'name')** **tbml('property', '-name', 'REST', 'ipAddressV4')** returns the value of the **name** property for the resource with ID **resource_0**. In the example, returns **myRouter** **Note** In Python, list unnamed arguments first and then a named argument in any order. An unnamed argument cannot be listed after a named.argument. E.g., name=value arguments can only be at the end of a procedure call. For e.g., my_procedure(arg1, arg2, ..., argN, kwArg1=val1, ... k2ArgN=valN)
tbml property -name *namePath propertyPath* **tbml('property','-name'**, *'namePath', 'propertyPath')*	Returns the value of the property specified by *propertyPath* for the resource or link specified by *namePath.* **Example** **tbml property -name "myRouter card1" [list "my property collection" myCustomProperty]** **tbml('property', '-name', 'myRouter card1', 'myCustomProperty')** 1. First, finds the resource **card1**.We specified the full name path starting from the top level resource which is **myRouter**. 2. Then finds the value for **myCustomProperty**, which is a child of the property collection group called **my property collection**. In the example, returns the value **hello world**

tbml property	Returns property value, Id, etc., as specified
tbml property -displayName *DisplayNamePath propertyPath* **tbml('property', '-displayName',** '*DisplayNamePath', 'propertyPath')*	Returns the value of the property specified by *propertyPath* on the resource that is specified by *DisplayNamePath*. **Example** **tbml property -displayName "myRouter_displayName card1_displayName port1_displayName" ipAddressV4** **tbml('property', '-id', 'resource_0', '-displayName', 'card1_displayName port1_displayName', 'ipAddressV4']** 1. First, finds the resource **port1** (because we specified the resource or link full name path starting from the top-level resource **myRouter**). 2. Now that **port1** is found, finds the **ipAddressV4** property on port1. In the example, returns the value **10.100.2.3**
tbml property -id *ID* **-name** n*amePath propertyPath* **tbml('property', '-id',** '*ID'*, '**-name',** 'n*amePath', 'propertyPath')*	For a resource or link with ID of *ID*, returns the value of the property at *propertyPath.* *namePath* is the hierarchical path using the **name** property relative to the resource or link with the specified ID. *propertyPath* is the path to a property using the name of the property. **Example** **tbml property -id resource_0 -name "card1 port1" ipAddressV4** **tbml('property', '-id', 'resource_0', '-name', 'card1 port1', 'ipAddressV4')** 1. First, finds the resource with ID **resource_0**. 2. Next, finds a child resource of **resource_0** that has the name **card1**. 3. Then finds a child resource of **card1** that has name **port1**. 4. Now that **port1** is found, finds the **ipAddressV4** property for **port1**. In the example, returns the value **10.100.2.3**

tbml property	Returns property value, Id, etc., as specified
tbml property -id *ID* **-displayName** *displayNamePath propertyPath* **tbml('property', '-id**', '*ID*' '**-displayName', '***displayNamePath', 'propertyPath')*	For a resource or link with ID of *ID* and for a property on the resource with a **displayName** of *DisplayNamePath*, returns the value of the property specified by *propertyPath.* *displayNamePath* is the hierarchical path using **displayName** property relative to the resource or link with the specified ID **Example** **tbml property -id resourece_0 -displayName "card1_displayName" "port1_displayName" ipAddressV4** **tbml('property', '-id', 'resourece_0', '-displayName', '"card1_displayName"', '"port1_displayName"', 'ipAddressV4')** 1. First, finds the resource with ID **resource_0**. 2. Next, finds a child resource of **resource_0** that has a **displayName** of **card1_displayName**. 3. Then finds a child resource of **card1** that has a **displayName** of **port1_displayName**. 4. Now that **port1** is found, it finds the **ipAddressV4** property on port1 and returns the value **10.100.2.3**
tbml property *propertyPath* **tbml('property', '***propertyPath')*	Returns the value of the specified property for the topology itself (Expressed another way; returns the value of the specified property that appears in the tbml file's header element). **Examples** **[tbml property description]** **tbml('property', 'description')** Returns the value of the **description** property for the topology. **tbml property [list "my collection"] myProperty** In the example, returns the value of the **myProperty** property, which is a child of the property collection called "**my collection**".

tbml property	Returns property value, Id, etc., as specified
tbml property -vendorId?*vendorID*? <any other arguments> **tbml('property', '-vendorId',** '*vendorID',* 'any other arguments')	For all variations of the **property** subcommand, you can specify an optional **-vendorId** argument to find all properties with the vendor ID specified by *vendorID*. • If the *vendorID* value is specified and a property match exists but has a different vendor ID, then returns an empty list. • If no *vendorID* value is specified and multiple properties match, each with different or no vendor ID, then returns all matches. • If a *vendorID* value is specified and a property match exists but does not have a vendorID, then returns an empty list. • If the property is a member of a property collection, then, in the case that the collection includes other collections, the *vendorID* match test is applied only to the top-level property collection. In this example, if you specify **vendorA**, then **propertyC-vendorC** is returned even though it is a vendorC property.  **Example** **tbml property -vendorId com.spirent -id resource_0 author** **tbml('property', '-vendorId', 'com.spirent-id', 'resource_0', 'author')** Returns the values of all properties named **author** whose **vendorID** is set to **com.spirent** on the resource with **ID** of **resource_0**.
tbml property -default *defaultValue* <any other arguments> **tbml('property', '-default',** *'defaultValue',* 'any other arguments')	For all variations of the **property** subcommand, you can specify a default value to return in the case that no property value was found. If the **-default** option is not used and no property was found, iTest generates an **onInterpreterError** execution issue. **Example** **tbml property -default Sam -id resource_0 author** **tbml('property', '-default', 'Sam', '-id', 'resource_0', 'author')** Returns the values of all **author** properties on resource with ID **resource_0**. In the example, if no value is returned for the **author** property, returns **Sam**

deviceList subcommand: Return device ID

The **tbml deviceList** command returns a list of device IDs for the specified parent. You must specify the device's **ID**, **name**, or **displayName**.

deviceList supports a combination of ID, name path, and displayName path to determine the starting resource, (as described for the **property** subcommand in "property subcommand: Return value of a property" on page 526).

See "tbml command syntax" on page 524.

Issues

- If the resource was not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.

- The starting resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

tbml deviceList	**Returns a list of the top-level device IDs.**
tbml deviceList -name *namePath* **tbml('deviceList',** '**-name'**, '*namePath')*	Returns device IDs in a list for the specified device. **Example** **tbml deviceList -name "myRouter card1"** **tbml('deviceList', '-name', 'myRouter card1')** Finds immediate children of card1. In the example, returns **resource_0_0_0** (the ID for port1)
tbml deviceList -displayName *displayNamePath* **tbml('deviceList',** '**-displayName'**, '*displayNamePath')*	Returns device IDs in a list for the specified device. **Example** **tbml deviceList -name "myRouter_displayName card1_displayName"** **tbml('deviceList', '-name', 'myRouter_displayName', 'card1_displayName')** Finds immediate children of **card1** and returns their IDs. In the example, returns **resource_0_0_0**(the ID for port1)
tbml deviceList -id *ID* **tbml('deviceList',** '**-id',** '*ID')*	Returns device IDs in a list for the specified device. **Example** **tbml deviceList -id resource_0** **tbml('deviceList', '-id', 'resource_0')** Finds immediate children of **resource_0**. In the example, returns **resource_0_0** (the ID for **card1**)

tbml deviceList	**Returns a list of the top-level device IDs.**
tbml deviceList -id *ID* **-name** *namePath* **tbml('deviceList', '-id',** '*ID*', '**-name',** '*namePath')*	Returns device IDs in a list for the specified device. **Example** **tbml deviceList -id resource_0 -name card1** **tbml('deviceList', '-id', 'resource_0', '-name', 'card1')** Finds immediate children of **card1** and returns their IDs. In the example, returns **resource_0_0_0** (the ID for port1)
tbml deviceList -id *ID* **-displayName** *displayNamePath* **tbml('deviceList', '-id',** '*ID*', '**-displayName',** '*displayNamePath')*	Returns device IDs in a list for the specified device. **Example** **tbml deviceList -id resource_0 -displayName card1_displayName** **tbml('deviceList', '-id', 'resource_0', '-displayName', 'card1_displayName')** Finds immediate children of **card1** and returns their IDs. In the example, returns **resource_0_0_0** (the ID for port1)

query subcommand: Return XPath query result

tbml XPath query	**Returns valid XPath query.**
tbml query *XPath* **tbml('query',** '*XPath')*	Returns the value returned by the specified valid XPath query. **Example** **tbml query //tbml/body/resources/resource/@id]** **tbml('query', '//tbml/body/resources/resource/@id')** returns IDs for all top-level devices **tbml query //tbml/body/resources/resource\[@id="resource_0"\]/property\[@name="ipAddressV4"\]** returns the ipAddressV4 property value on a resource with ID **resource_0**. **Note** You must use the backslash character to escape square brackets in the XPath expression.

endpoint subcommand: Return resource IDs of two endpoint resources

Returns the resource IDs of the two endpoint resources that are directly connected to the specified link. If the link is connected to a port on a device, then the resource ID of the port (not of the device) is returned.

See "tbml command syntax" on page 524.

Issues

- If the link is not found, iTest generates an **onInterpreterError** execution issue.
- The starting link should be unique. If multiple links are found, iTest generates an **onInterpreterError** execution issue.

tbml endpoint	**Returns TBML endpoint values as specified.**
tbml endpoint -name *name* **tbml('endpoint', '-name',** *'name')*	Returns the two endpoint resource IDs in a list for any link object given its **name**. Multiple matches are found if multiple links have the same **name**. **Note** *namePath* argument is not supported because link elements cannot be hierarchical. **Example** **tbml endpoint -name upstream** **tbml('endpoint', '-name', 'upstream')** Returns **resource_0** and **resource_1** because the two resources are at either end of the link with a **name** of **upstream**.
tbml endpoint -displayName *displayName* **tbml('endpoint', '-displayName',** *'displayName')*	Returns the two endpoint resource IDs in a list for the link object specified by its **displayName**. Multiple matches are found if multiple links have the same **displayName**. **Note** *displayNamePath* argument is not supported because link elements cannot be hierarchical. **Example** **tbml endpoint -displayName upstream_displayName** **tbml('endpoint', '-displayName', 'upstream_displayName')** Returns **resource_0** and **resource_1** if the two resources are at either end of the link with a **displayName** of **upstream_displayName**.
tbml endpoint -id *ID* **tbml('endpoint', '-id',** *'ID')*	Returns the two endpoint resource IDs in a list for the link object that is specified by *ID*. **Note** The *namePath* and *displayNamePath* arguments are not supported because link elements cannot be hierarchical. **Example** **tbml endpoint -id link_0** **tbml('endpoint', '-id', 'link_0')** Returns **resource_0** and **resource_1** because the two resources are at either end of the link.

parentList subcommand: Return list of parent IDs for a resource

The **parentList** subcommand returns the list of parent IDs for a specified resource.

Parents are listed starting from the root and continuing down the chain to the immediate parent. For a root resource, **parentList** returns an empty list.

parentList supports a combination of ID, name path, and displayName path to determine the starting resource, (as described for the **property** subcommand in "property subcommand: Return value of a property" on page 526).

See "tbml command syntax" on page 524

Issues

- If the resource was not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.
- The starting resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

tbml parentList	**Returns TBML parentList values as specified.**
tbml parentList -name *namePath* **tbml('parentList', '-name', '***namePath')*	**Example** **tbml parentList -name "myRouter card1 port1"** **tbml('parentList', '-name', 'myRouter', 'card1', 'port1')** Returns **resource_0** and **resource_0_0** because the specified resource is for a port, and its parents are **myRouter** whose ID is **resource_0** and **card1** whose ID is **resource_0_0**
tbml parentList -displayName *displayNamePath* **tbml('parentList', '-displayName',** '*displayNamePath')*	**Example** **tbml parentList -displayName "myRouter_displayName card1_displayName"** **tbml('parentList', '-displayName', 'myRouter_displayName card1_displayName')** Returns **resource_0** because the specified ID is for a card, and its parent is **myRouter** whose ID is **resource_0**
tbml parentList -id *ID* **tbml('parentList', '-id',** *'ID')*	**Examples** **tbml parentList -id resource_0** **tbml('parentList', '-id', 'resource_0')** Returns an empty list because **resource_0** is a top level resource. **tbml parentList -id resource_0_0_0** returns **resource_0** and **resource_0_0** because the specified resource is for a port, and its parents are **myRouter** whose ID is **resource_0** and **card1** whose ID is **resource_0_0**

tbml parentList	Returns TBML parentList values as specified.
tbml parentList -id *ID* **-name** *namePath* **tbml('parentList', '-id',** *'ID',* **'-name',** *'namePath')*	**Example** **tbml parentList id resource_0 -name card1** **tbml('parentList', 'id', 'resource_0', '-name', 'card1')** Returns **resource_0** because the specified ID is for **myRouter**, which contains a child resource named **card1**, and its parent is **myRouter** (whose ID is **resource_0**).
tbml parentList -id *ID* **-displayName** *displayNamePath* **tbml('parentList', '-id',** *'ID',* **'-displayName',** *'displayNamePath')*	**Example** **tbml parentList id resource_0 -displayName "card1_displayName port1_displayName"** **tbml('parentList', 'id', 'resource_0', '-displayName', 'card1_displayName port1_displayName')** Returns **resource_0** and **resource_0_0** because the specified resource is for a port, and its parents are **myRouter** whose ID is **resource_0** and **card1** whose ID is **resource_0_0**

linkList subcommand: Returns list of IDs of link objects

The **linkList** subcommand returns the list of IDs of all link objects that are a link for the specified resource or children of the specified resource. Returns an empty list if the specified resource or its children do not have any links to other resources.

linkList also supports a combination of ID, name path, and displayName path to determine the starting resource, (as described for the **property** subcommand in "property subcommand: Return value of a property" on page 526).

See "tbml command syntax" on page 524

Issues

- In the event that the resource was not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.
- The starting resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

tbml linkList	Returns TBML linkList values as specified.
tbml linkList -id *ID* **tbml('linkList', '-id',** *'ID')*	**Example** **tbml linkList -id resource_0** **tbml('linkList -id', 'resource_0')** returns **link_0** and **link_1** because **resource_0** contains **card1** which is connected using **link_0**, and **card1** contains **port1** which is connected to something using **link_1**
tbml linkList -name *namePath* **tbml('linkList', '-name',** *'namePath')*	**Example** **tbml linkList -name "myRouter card1"** **tbml('linkList', '-name', 'myRouter', 'card1')** Returns **link_0** and **link_1** because **card1** is connected using **link_0**, and **card1** contains **port1** which is connected using **link_1**

tbml linkList	**Returns TBML linkList values as specified.**
tbml linkList -displayName *displaynamePath* **tbml('linkList', '-displayName',** '*displaynamePath')*	**Example** **tbml linkList -displayName myRouter_displayName** **tbml('linkList', '-displayName', 'myRouter_displayName')** Returns **link_0** and **link_1** because **myRouter** contains **card1** which is connected using **link_0**, and **card1** contains **port1** which is connected using **link_1**
tbml linkList -id *ID* **-name** *nameList* **tbml('linkList', '-id',** '*ID',* '**-name',** '*nameList')*	**Example** **tbml linkList -id myRouter -name "card1 port1"** **tbml('linkList', '-id', 'myRouter', '-name', 'card1', 'port1')** Returns **link_1** because **card1** contains **port1** which is connected using **link_1**
tbml linkList -id *ID* **-displayName** *displayNamePath* **tbml('linkList', '-id',** *'ID',* '**-displayName',** '*displayNamePath')*	**Example** **tbml linkList -id myRouter -displayName "card1_displayName port1_displayName"** **tbml('linkList', '-id', 'myRouter', '-displayName', 'card1_displayName port1_displayName')** Returns **link_1** because **card1** contains **port1** which is connected using **link_1**

sessionList subcommand: Return list of session names for a device

For the resource that you specify using its device ID, the **tbml sessionList** command returns the list of session names attached to the resource (from the session profiles and from the sessions defined in the topology). See "tbml command syntax" on page 524.

Tip To determine a device ID, use the **tbml deviceList** command as described in "deviceList subcommand: Return device ID" on page 531.

Issues

- If the resource was not found or if multiple resources are found for the specified device, iTest generates an **onInterpreterError** execution issue.

tbml sessionList -id *ID* **tbml('sessionList', '-id',** '*ID')*	Returns session names in a list for the specified device ID. **Example** **tbml sessionList -id resource_0** **tbml('sessionList', '-id', 'resource_0')** Returns names of all session profiles attached to **resource_0**

remoteEndpoint subcommand: Return the resource ID of connected resource

The **remoteEndpoint** subcommand returns the resource ID of the resource that is immediately connected to the specified resource. **remoteEndpoint** returns multiple matches if the specified resource is connected to multiple resources.

remoteEndpoint also supports a combination of ID, name path, and displayName path to determine the starting resource, (as described for the **property** subcommand in "property subcommand: Return value of a property" on page 526).

See "tbml command syntax" on page 524.

Issues

- If the resource is not found for the specified ID, name, or displayName, iTest generates an **onInterpreterError** execution issue.
- The starting resource should be unique. If multiple resources are found, iTest generates an **onInterpreterError** execution issue.

tbml remoteEndpoint	**Returns TBML remoteEndpointvalues as specified.**
tbml remoteEndpoint -name *namePath* **tbml**('**remoteEndpoint', '-name',** '*namePath')*	**Example** **tbml remoteEndpoint -name "myRouter card1"** **tbml('remoteEndpoint', '-name', 'myRouter card1')** Returns **resource_1_0** because **card1** is connected to **card1** on **mySwitch**
tbml remoteEndpoint -displayName *displayNamePath* **tbml**('**remoteEndpoint', '-displayName',** '*displayNamePath')*	**Example** **tbml remoteEndpoint -displayName "myRouter_displayName card1_displayName"** **tbml('remoteEndpoint', '-DisplayName', 'myRouter_displayName card1_displayName')** Returns **resource_1_0** because card1 is connected to card1 on **mySwitch**
tbml remoteEndpoint -id *ID* **tbml**('**remoteEndpoint', '-id', '***ID')*	**Example** **tbml remoteEndpoint -id resource_0_0** **tbml('remoteEndpoint', '-id', 'resource_0_0')** Returns **resource_1_0** because **card1** is connected to **card1** on **mySwitch**

tbml remoteEndpoint	**Returns TBML remoteEndpointvalues as specified.**
tbml remoteEndpoint -id *ID* **-name** *namePath* **tbml**(‘**remoteEndpoint’, ‘-id’,** ‘*ID’,* **‘-name’,** ‘*namePath’)*	**Example** **tbml remoteEndpoint -id resource_0 -name card1** **tbml(‘remoteEndpoint’, ‘-id’, ‘resource_0’, ‘-name’, ‘card1’)** Returns **resource_1_0** because **card1** is connected to **card1** on **mySwitch**
tbml remoteEndpoint -id *ID* **-displayName** *displayNamePath* **tbml**(‘**remoteEndpoint’, ‘-id**’, ‘*ID’* **‘-displayName’,** *‘displayNamePath’)*	**Example** **tbml remoteEndpoint -id resource_0 -displayName card1_displayName** **tbml(‘remoteEndpoint’, ‘-id’, ‘resource_0’, ‘-displayName’, ‘card1_displayName’)** Returns **resource_1_0** because **card1** is connected to **card1** on **mySwitch**

Global Topology

Global topology

If you identify a particular topology as the ***Global*** topology, then, when you execute any test case that does not specify a **Local topology**, iTest asks whether you would like to use the Global topology for execution. You can configure the behavior so that particular test cases ignore the Global topology while others use it. (For more information on Local topologies, see "Test Case editor: General page" on page 162.)

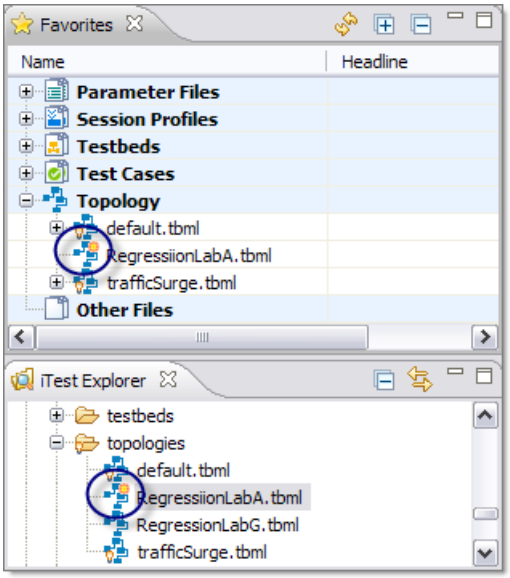
The topology that was used for execution is identified in the Execution view and in test reports.

Specifying a topology as the Global topology

In the iTest Explorer or Favorites view, right-click the topology and select **Set As Global Topology**.

To undo the setting, right-click the Global topology and select **Clear Global Topology** or specify that another topology is the Global topology.

The icon for the Global topology is marked to distinguish it from other topologies:



Topology used during execution

You have the option to modify the default action noted here by using the **Preferences** page to configure iTest to use either topology. See "Setting preferences for execution" on page 281.

Global topology specified?	Local topology specified?	Topology used during execution
No	No	None
Yes	No	The default action is to display a dialog box that asks for your choice.

Yes	Yes	The default action is to display a dialog box that asks for your choice.
No	Yes	Local

Specifying topologies when executing itestRT (headless execution)

When you execute a test case using a headless version of iTest, you can use a flag to pass in the topology to use. The specified topology overrides the **Local topology** that is configured in the test case and the Global topology.

See Chapter 32, “iTest Runtime: iTestRT”.

Topology Preferences

Set preferences for Topologies

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Topologies**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Topologies > Palette

Palette configuration file	You can edit the default file as needed (for example, to add or modify device definitions) or specify a different file. Default: project://resources/topology/palette.xml

Spirent > Topologies > Print

Specify the common print settings for printing topology image files.

Spirent > Topologies > Ruler and Grid

Specify the common ruler and grid settings for displaying topology elements on the canvas.

CHAPTER 26

Response Maps: Returning Data from Responses

Response maps

A text response from a session is either structured or unstructured. If a response is structured, it contains a native format that identifies the data in the response. Examples of structured responses are HTML, XML, TL-1, JSON, etc. – all formats that use tags and formatting to identify data. iTest natively identifies structured responses and can easily create accurate iTest queries for the data located within structured responses.

Unstructured responses are text responses with no parsing format. There is no native formatting that identifies the location of data in the response. These are the most common type of responses you will see in iTest. For example, a response table may look neat and orderly to us humans and therefore seem to be structured. Because it is purely a text file with tabs or spaces to improve its appearance and has no XML structure, however, it is ***not*** a structured response and no computer program can generated ***guaranteed*** queries to return particular items of data from the response. (iTest can do a pretty good job of guessing, however!)

You can think of a response map as a template that you define to bring structure to an unstructured response. The template specifically identifies the data within the response with queries that can return particular values from the response. When a response map is applied to a response, you will see each item of data in color (in the Response view) that matches the queries defined in the response map.

Typically, you use a response map to return the data that you are interested in from the response to a step in a test case. There are other uses that we will discuss later.

Overview: Response map document

- Defines one or more mappers that describe how to return data from the unstructured text response. Mappers also describe how to query information from the structured data part of the response. For example, a mapper can identify the timestamp in a response and can structure a table of values so that a query can return the value of dropped bits so that an analysis rule can decide whether to pass or fail the test case.
- Contains one or more sample responses to a particular command
- Can hold a sample response that a session can use as an emulated response (as if from a “virtual testbed”)

Use the Response Map editor to create and edit response maps. As you add new parts to the map definition, you may see new editor pages appear.

The editor works in conjunction with other views including the Response, Structure, Queries, and Step Issues views. While you are working in the editor, the other views provide

auto-updated feedback on how the response map will operate on the response sample that is currently selected on the **Samples** page of the editor.

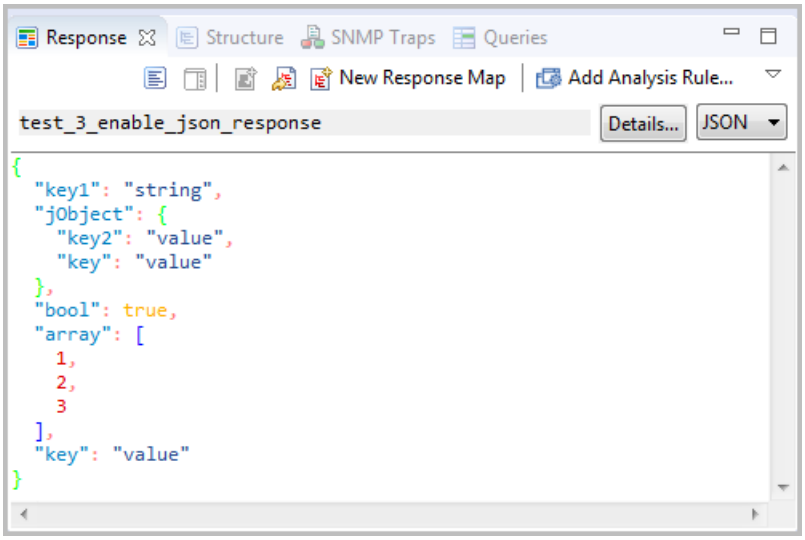
Why create a response map?

When iTest captures a response, the response text appears in the Response view and, in addition, iTest creates a structured data (XML formatted) version of the response. There are two distinct uses for response maps:

A response map can reliably return data from a response for analysis or for other uses

When you create a response map, you create queries that can reliably return particular values from any response to the particular command (based on the structured sample response that you used to create the response map). As a result, when you later execute a test case, an analysis rule for a step can use a query defined in the map to return values from a response and can then evaluate the response to determine pass/fail for the test case or to perform other tasks.

In this example response, the matches to queries (defined in the response map) are in color on a white background— your visual cue that the mapper has done its work and returned all matches to defined queries. The values in quotes are available for you to work with using analysis rules.



iTest automatically maps queries for keys at the root level of a JSON object response. There is no need to create auto-mapping queries for nested keys.

So, a response map is a more-or-less complete structuring of a response so that any value of any of the data fields of interest can be returned for use by the test case developer. A map is a general-purpose tool that you can reuse in different test cases for different testing / analysis purposes.

The process of response mapping has two main purposes:

- Parsing an unstructured response and returning the interesting data into a structured version of the response.

- Providing meaningful queries to mine the data in that structured portion of the response. The process of mapping is performed either by built-in mappers for some session types (Web, SNMP, TL1, JSon, all of the traffic generator session types) or by passing a text response (for example, from Telnet or SSH) through a response map.

A response map can store sample responses for virtual testbeds (emulation)

One of the options for the virtual testbed (emulation) feature is that you can specify an ***external source*** for the emulated response to a step. You specify the source response as a particular sample that is stored in a response map or response map library.

For a full discussion of implementing a virtual testbed by emulating responses, see Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions". For instructions on creating a response map for this purpose, see "Creating a response map: Instructions" on page 554.

Overview: Creating a response map

This overview describes the process of creating a map for the most typical case — using the map to return particular data from responses during execution. The additional function of response maps — storing a sample response for use as an emulated response (as if from a "virtual testbed") — is described in the detailed instructions.

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Step 1A: Get a sample response

A response map normally contains one or more samples of the responses that it is intended to map. Before you create a response map, find a good sample response to use while creating the map. The most common way to get a sample is to start with a test case containing a step that produces the type of response that you want to map. Select the step in the Test Case editor and the Response view displays the most recent response received while executing that step. You can also populate the Response view when you select a step in a Test Report or in the Capture view.

Step 1B: Optional: Filter the response if needed

The Response Filtering feature enables you to remove unwanted text from a response after a step has completed and before iTest applies analysis rules. The resulting portion of the response is cleaner to read, to understand, and to map. This feature is useful in the following situations:

- The response contains a lot of text, but it would be easier to analyze and display a portion of the response and to ignore the rest.
- The device produces logging messages that are mixed into the output (common for TL1). The messages appear as separate lines. You might want to analyze the messages in a different step, but you need to filter out the messages so that you can define a response map for the base response. (More information on working with TL1 responses appears in Chapter 61, "TL1 Sessions".)
- The device produces XML output, but the output includes non-XML headers and footers that mess up the XML (or HTML) mapping. Filtering can remove the headers and footers before you apply the queries.

Step 2: Choose a location for the response map

Before you actually create a response map, you should first think about where the map belongs in your workspace. In most cases, you will want to store the response map in a response map library that contains all maps for a particular device type. In special cases, you might want to create your own response map library that contains your own maps, but also inherits maps from another shared response map library. In other cases, you may want to store the response map along with the test case – in those cases where the response map is only useful in that one special case.

Step 3: Create the map

A response map is a file. The easiest (and recommended) way to create a new response map is to start from the **Response** view when it contains a good sample of the response. Click  **New Response Map** to start the **Response Map** wizard. The wizard will pre-populate the map based on information you provide.

If you have the Automatic Response Mapping feature installed, then you can also ask it to analyze the sample and try to automatically construct an appropriate map for you. When the wizard finishes, the new response map file will have been created, and will open in the Response Map editor so you can proceed with your work.

Limitations of the Auto-mapping feature

- Auto-mapping does not find tables where the row count is less than or equal to 2.
- Auto-mapping does not process responses larger than 250 lines.
- Auto-mapping can become confused by table rows that wrap onto a newline. This is a tricky problem to map manually as well. We recommend that you increase the terminal width so that response does not wrap on a new line.

An alternative is to create a new response map from scratch. You select **File > New > iTest > Response Map** from the main menu. This will create a new response map file at the location you request but will not pre-populate it in any way.

Step 4: Choose mapping technology

iTest response maps can use different mapping technologies to return structured data from the unstructured textual response. There are three types of mapping currently supported: pattern-based, table-based, and block-based. Each of these technologies is optimized for different types of responses. You can combine these if your response contains different portions that are best mapped with different technologies.

Auto-mapping If you requested automatic response map generation in the Response Map Wizard, then you may be starting with a map that already contains one or more different mapping technologies. Otherwise, you need to select one of the mapping technologies from the General page of the response map editor. To decide which one to use, review the following:

Table-based mapping You will use table-based mapping if the response contains a classic table made up of rows and columns where each row is represented by a single line of text in the response. If the same table appears multiple times in the response, a single table map can handle that. If the response contains multiple tables with different structures, you can use multiple table map definitions in the same response map.

Pattern-based mapping You will use pattern-based mapping in most other cases (except as discussed below for block-based mapping). This technology uses regular expressions without requiring that you be an expert in regular expression syntax. You select a line or sequence of

lines in the sample response and identify the tokens within those lines that you want to return. The pattern mapper constructs a sequence of regular expressions for you.

Block-based mapping Block-based mapping is typically used only for the most complex responses. These responses typically have repeating and/or nested multiline structures within them. These structures may be slightly variable and/or optional. You want to return data and usually want to key it to some header information in these structures or "blocks". Block mapping is powerful, but requires some experience in order to create robust maps.

Structured mapping Responses that have well-defined structure (XML, TL1, JSON) are mapped by built-in mappers.

Step 5: Construct the map

Once you have chosen the appropriate response mapping technology, you need to populate the properties of the map accordingly. This will depend on which technology (or technologies) you have chosen to use.

In the response map editor, click the editor tab corresponding to the technology of interest — "Pattern", "Table", and so on. If you used automatic map generation, the page may already be populated with information that you can then further modify. If not (or automatic map generation was unable to help), then you need to populate that information yourself.

You should be in the iTest Response Mapping perspective. (If not, use the **Switch Perspective** button along the top to switch to this perspective.) The Response view should be populated with the sample contained in your response map. (If you have multiple samples, you can select a different one in the Samples page in the editor, and that one will then show up in the Response view.) To help you get started, there is a sidebar that may be visible that is like a mini-wizard that can help you get started with the map creation – depending on the type of technology you are using. For a table map, for example, as soon as you create a new table map definition (by clicking ⊞ in the Table editor page), you will see buttons in the sidebar of the Response view that will guide you through the process of identifying the table banner, footer, columns, and so on. For Pattern mapping, you will see a button that lets you create a new pattern using the lines selected in the Response view.

While these aids can be helpful, you will still need to understand how the mapping technology works so that you can customize it to your own situation. The maps contain many configurable properties that allow you to tightly control how data will be returned.

Step 6: Add custom queries (optional)

A response map helps to transform unstructured data (in the text response) into structured data (as XML). It also defines a set of queries that can be applied to the structured data to return the data of interest. The Structure view shows all of the structured data that goes along with the response that is shown in the Response view. The Queries view shows a list of predefined queries that can be applied to the structured data. When analysis rules use the "query" extractor, they are applying a query to the structured data to get the information to be returned and analyzed. Any valid XPath query can be used in the analysis rule's query extractor. But using predefined queries is easier because you can pick them from a list.

About the "blue boxes" The blue boxes that surround certain data in the Response view correspond to data that will be extracted by a predefined query. Clicking one of the boxes and adding an analysis rule is just a another method for selecting the corresponding query in the Queries view and adding an analysis rule from the view.

When you create a response map, it will automatically construct a set of predefined queries that naturally go with the map. For example, on a table map, it will construct queries that extract cell values — possibly based on a key column if one has been defined. As a response map designer, you may find that it is very helpful to the users of your map to provide additional queries beyond what the map creates automatically for you. For example, you may want to create queries that provide meta-data about the response – such as a count of the total number of rows in your table, or the sum of the values of one of the columns in your table. Or you might want to perform arithmetic on certain values to produce another meta-value. All of these things are possible via custom queries.

To add custom queries, choose the Queries tab in the response map editor (along the bottom of the editor) and you can add your own custom queries.

- **Step 7: Verify the map**

  Before you are finished, you should make sure that your response map works as you intended.

  When you have your response map open in an editor, the Step Issues view will show you any response mapping problems encountered when mapping the sample(s) associated with that map. You should resolve all of these issues. It is common for a response map to work properly with one sample response, but fails for another sample. So it is a good idea to find two or three different samples of the response that the map is intended for. You can add these samples on the Samples page in the response map editor, and the Step Issues window will show you if there are any problems mapping against any of the samples.

  The Queries view will show you a list of all of the predefined queries associated with the current response map as well as the result of applying that query to the selected response sample. This is the list that users will see when trying to add analysis rules to a step that uses this response map. So make sure that this list contains all of the queries that you would want a user to have access to. Make sure the names are meaningful to other users.

  The Response view will show the sample response with blue boxes around all data in the response that has been properly mapped. If there are missing blue boxes, these should be investigated and resolved.

- **Step 8: Use the map**

  If your response map has been stored in a response map library, and you have configured the appropriate applicability data, then your response map should be automatically associated with steps in test cases accordingly. You should check that this is working properly.

  In other cases, you will need to associate the response map with the step explicitly. You do this on the Expected Response page under “Other Post-processing” within the step properties in the test case editor.

  We recommend that you associate response maps with test case procedures. This is a nice way to provide “blue boxes” for the text returned in a reusable procedure. You associate a response map with the procedure using the Response Map property on the Procedure property page in the test case editor.

Response mapping tips

- If you need to return only a small number of values from a response, it is probably faster and easier to add individual queries rather than creating a response map. You can do that

from the Response view, the Queries view, or the Structure view. In the view, select the value, right-click it, and then select **Add Rule**.

- Once you create a response map, you can access it quickly from a test case by right-clicking the appropriate step and then selecting **Open Response Map**.
- For ease of maintenance and understanding, you typically name a response map with the text of the command that results in the response (for example, name the response map for the **show interfaces** command **show_interfaces**).
- For very large responses, allocate more memory to iTest. See "Specifying how much memory to allocate to iTest" on page 69.

Choosing a mapping technology

This topic describes how to determine which of iTest's mapping technologies to apply to a particular response.

Mapping is the process of filling in the structured (XML) portion of the response to a step in a test case (This is the process that results in the blue boxes in the Response view that alert you that it is easy to query a response for the value.) Some session types fill this structured data in all by themselves (Web, SNMP and exec). Some session types do not auto-fill their structured data, but do have a well defined structure (for example, XML, HTML, JSON, and TL1).

iTest's six built-in mappers (XML, HTML, TL1, Pattern, Table, and Block) are optimized for filling in the structured data portion of text responses. XML, HTML, JSON, and TL1 do not require any configuration, nor do they require a response map file. They will be automatically invoked based on the content-type setting of the textual response.

For an unstructured response, you can use Pattern, Table, or Block mappers to extract key pieces of data (often called tokens) into the structured data. For the most part a "token" is a piece of text from the response that gets put into an XML element in the structured data. Tokens generally do not span lines. Tokens are indicated by blue boxes.

Choosing the correct mapping technology will simply the process of developing a good response map for a particular response. The first thing to understand is that Pattern and Table can be used together and can map just a section of a response while ignoring the rest. Bock mapping by contrast, wants to consume the whole response. Most response maps from 2.x are Block, because that was the first one we had, but many of these would be easier to do using Table or Pattern.

Tip For very large responses, allocate more memory to iTest. See "Specifying how much memory to allocate to iTest" on page 69.

Pattern mapping

Pattern mapping is great at picking out that one piece of interesting data from a very unstructured response. To create a Pattern map, you copy the text you're interested in, including some surrounding "anchor" text into the **Identifying text** field. This works well when a label and its value appear together on a line, for example, **`inPackets = 344`**.

Table mapping

Often data is presented in tables with column headings. The Table mapper is optimized for returning data from text tables. Table mapping can be surprisingly versatile, and we will provide some examples of things that may not look like tables at first, but that you can use table mapping on.

Block mapping

Block mapping is a sophisticated way to match blocks of text within a response and their relationship to one another. Block maps are generally a little more rigid and care must be taken to configure blocks for all possible variations of a response. Block maps that consist of a single block are often the quickest and easiest way to map a simple response. On the other end of the spectrum, sophisticated responses with many repeating blocks of data are also well suited for block mapping.

Important Note the structured responses: HTML, XML, TL1, JSON, SNMP, Web, and traffic generator device responses: For HTML and XML responses, the structured data is auto-generated, so there is no need to apply one of the mapping technologies. iTest auto-maps TL1, SNMP, Web, and traffic generator device responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the generated queries.

Creating a response map: Instructions

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

---

Note One of the options for the virtual testbed (emulation) feature is that you can specify an external source for the emulated response to a step or session. You specify the source response as a particular sample in a response map. If you plan to use a response map solely to supply an emulated response, then the process of creating the response map document is simpler. The following instructions include notes on options you can skip.

Use the Response Map wizard to create a response map based on the current command/response pair. When you create a response map, you create queries that can reliably return particular values from any response (based on the structured sample response that you used to create the response map).

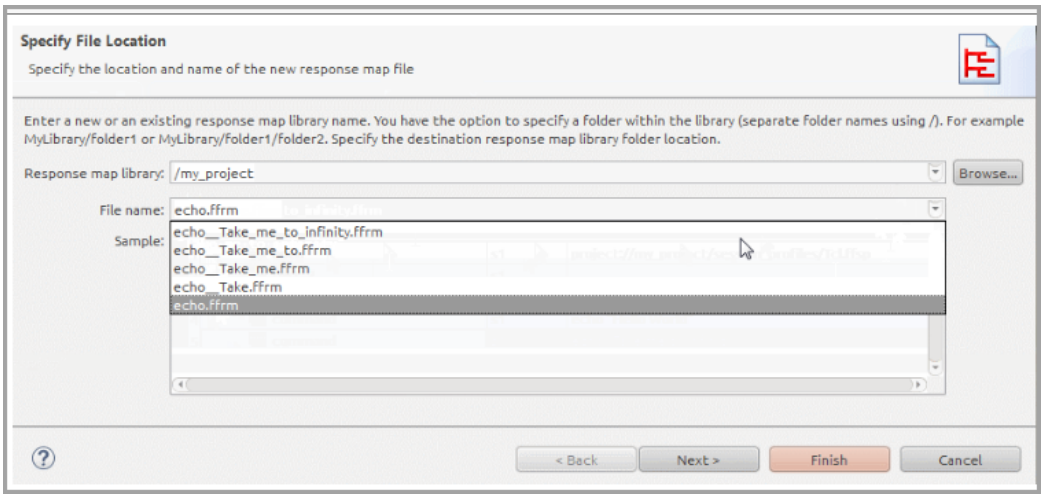
1 When you manually submit a command or a test case executes a command, the command/response pair appears in the Response view. If a response looks “good” (that is, it is in the format that you expect other users will see), you can click **Add Response Map** to start the Response Map wizard.

   Alternatively, in the Capture view, select the step that generates the response. Right-click and select **New Response Map**.

   Note If you need to filter the response text to make it easier to map (as described in “Step 1B: Optional: Filter the response if needed” on page 549), follow the instructions in Chapter 25, “Filtering Unwanted Text from Responses”.

   ---

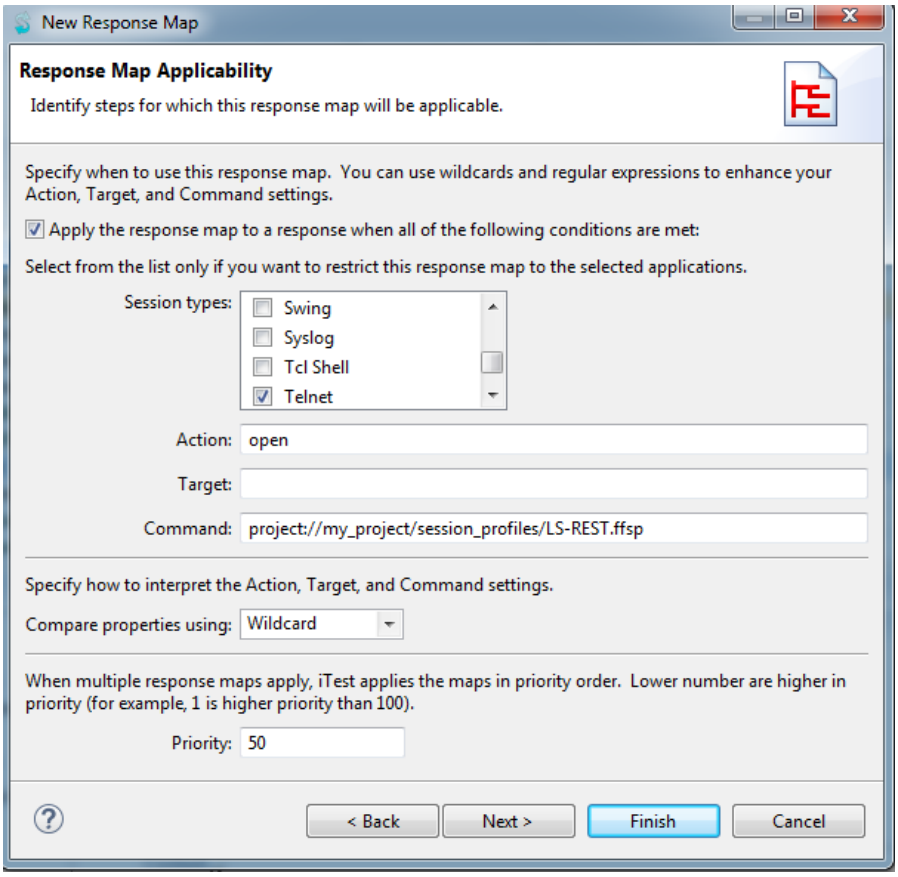
2 The **New Response Map** wizard starts and opens the **Specify File Location** page. On the **Specify File Location** page, you may modify the default values and specify the following properties for the new map. Click **Next.** .



Response map library	iTest discovers the project in which the new response map is created (based on the location of the test case), and selects the response_maps folder within that project (if it exists), as the default location. If you chose to add a new response map library, then specify a name for the new library. The new library will be added to the project and will appear in the iTest Explorer. If you chose to add the map to an existing response map library, then select the library from the list.
File name	iTest proposes multiple response map file names based on the command sent (command that produced the response). The multiple file names are suggested as a drop-down list, from most specific to least specific tokens within the command (contents of the file). See the example screenshot above. Select a name from the drop-down list or specify a name for the new response map. We recommend that you select a name from the list, e.g., the default name, which is the text of the command (with underscore characters for spaces).
Sample	Populated with the new response text.

3 On the **Response Map Applicability** page, specify when the map can be used. The settings on this wizard page are identical to the **Applicability** page on the Response Map editor (as

described in “Response Map editor: Applicability page” on page 567). Configure the settings and then click **Next**.



4 Use the **Automatic Response Map Generation** page to have iTest use automatic response mappers to take a first try at generating a map.

Important If you plan to use the response map only to supply an emulated response, then uncheck this option and click **Finish**.

Note To use this feature, you must check out an **Automatic Response Map Generation** license. To check out a license, click **Help > Configure iTest Licensing**.

In many cases the process generates a map and queries that you can use immediately. In some cases, you may have to rename particular queries. In other cases, the map will not meet your needs and you will have to generate a map using the Response Map editor.

Important Because we are continuously improving the auto-mapping and auto-query technologies, the maps and queries that it generates will change from one release of iTest to the next. For this reason, do not use the auto-mapping feature in test case steps to generate queries. Instead, use auto-mapping to generate a map and then save the resulting response map. In the test case, use the response map instead of the auto-mapping feature to obtain data from the response. This ensures that the test case will continue to work as you upgrade iTest.

5 Click **Finish**.

The wizard will auto check whether the session profile links to the response map library (project) specified and displays a dialog asking you to confirm whether you wish to configure the session profile to use the project specified as its response map library.

iTest places the new response map file into the specified response map library and then opens the response map file in the **Response Map** editor. You will now configure mappers and continue configuring the response map. See "Response Map editor: Overview page" on page 562.

If you plan to use the response map only to supply an emulated response, then you will not configure mappers. Open the editor's **Samples** page and continue as described in "Response Map editor: Samples page" on page 563.

Adding a sample response to an existing response map

On the Response view, click **Add This Sample to an Existing Response Map**  to quickly add a sample response to a response map. See "Adding additional samples to a response map document" on page 566.

XPath: Quick Overview

XPath is the foundation of response mapping. The iTest mapping process applies XPath queries to structured (XML) versions of the responses from sessions and match with particular data. When you view the data in blue boxes in the Response view, you are looking at the data in the response that matched with the XPath queries defined in the response map.

For each response from a session, iTest generates a *structured part* of the response — the response in XML format. Once the response is available in structured form, iTest can apply the XPath queries to match data in the response.

Here's an example of how an XPath query works. For the `show clock` command a response could be:

```
18:02:23.646 UTC Wed May 12 1993
```

iTest generates the following XML representation of the response (this is "structured data" — the structured part of the response):

```
<?xml version="1.0" encoding="utf-16"?>
<MapInformation ResponseMap="show_clock">
    <Details>
        <Body>
            <block>
                <date line="0" startcol="1" endcol="32">18:02:23.646 UTC Wed
May 12 1993</date>
            </block>
        </Body>
    </Details>
    <Regex />
    <Tabular />
</MapInformation>
```

In the example, the field (token) of interest is named **date**. We know that it is named **date** because the XML `<date>` tag surrounds the data that makes up the token. The XPath query used to reference the **date** token would be:

//block/date

and the query matches everything between the `<date>` and `</date>` tags. As a result, the text is surrounded by a blue box in the Response view to indicate the match to you. You can use analysis rules to work with the matching data.

Tip A *Query name* (sometimes called an *alias*) is a friendly name for an XPath query. Query names make advanced or conditional queries easier to read and more flexible. For example, we might give the name **timestamp** to the **//block/date** query.

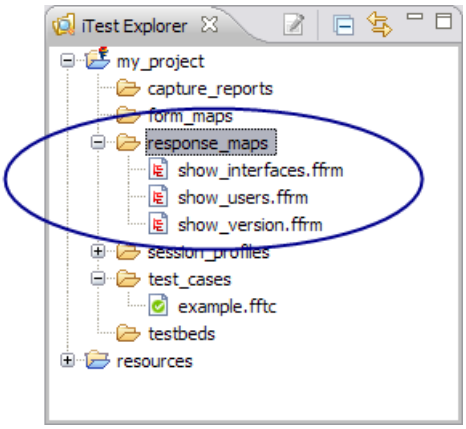
Response map libraries

When you create a new response map, the wizard asks you to specify the response map library to place the map into. The benefit of grouping associated response maps in libraries is that you avoid having to specify which response map to use when postprocessing a response.

- While you're creating a test case step, you can specify a particular response map to apply to the response during execution. Alternatively, you can specify that iTest should check through a library of response maps to determine which map to use to return the data.
- When you're defining a session profile, you can associate a response map library with the profile; so that any session that uses the profile will access the library when searching for applicable response maps for a step associated with the session.

Because each response map in the library has applicability settings, iTest can determine which maps apply and then apply the maps in priority order. The first response map that successfully maps the response is used to return the data.

A response map library is an iTest project (not a folder). A library contains a simple collection of response maps. Because maps are applied to a response in priority order as specified in the maps' applicability setting.



The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries. See "Making use of existing response map libraries: Chaining response maps" on page 559.

Making use of existing response map libraries: Chaining response maps

The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other specified libraries.

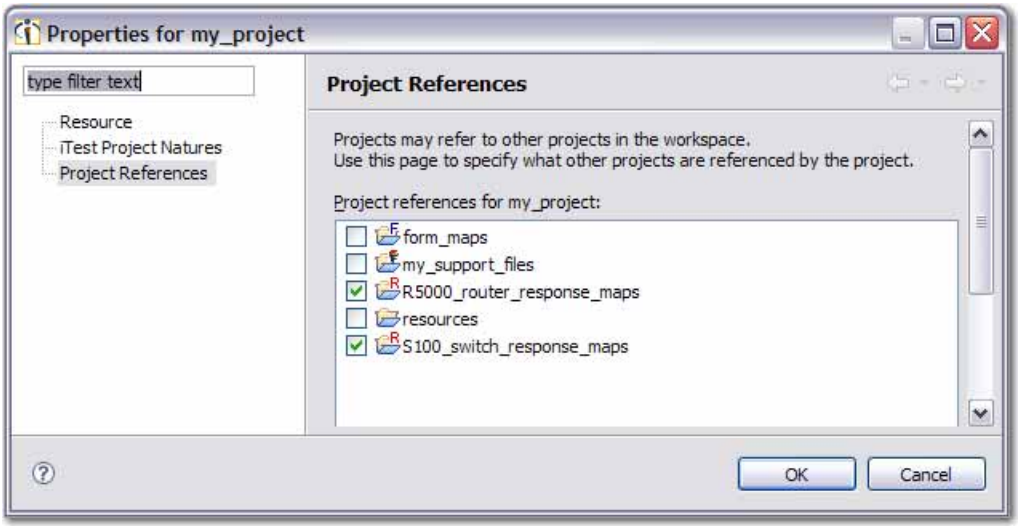
Response map chaining is helpful when:

- You do not have write-permission to modify the response maps in the corporate library, but need to provide alternative response maps for a step to try before (and in addition to) the response maps in the corporate library.
- You have sets of commands for device families and the specific devices (for example, one set for IOS and one for all 3K-series devices).

Note A custom parser is applicable only in the map in which it is used. Custom parsers are not imported from chained maps.

Chaining response map libraries

1. In the iTest Explorer, right-click the response map library that is specified for the session (the *primary* library) and select **Properties**. In the example, we right-clicked **my_project**.
2. Select **Project References** in the list.
3. On the **Project References** section, check the response map libraries that should be chained.
4. Click **OK**. Both **R5000_router_response_maps** and **S100_switch_response_maps** will now be chained to **my_project**.



Requirements for response map chaining

A response map is chained only when the following conditions are true:

- The step specifies a particular response map library. That is, for the step in the Test Case editor, the **Other Post-processing > Expected Response** property is set to **Use the response map library configured for the session**

- For the session profile associated with the step's session, a response map library is specified on the Session Profile editor's **Misc** page.
- The response map is applicable to the step. That is, on the Response Map editor's **Applicability** page, the settings result in the map being applicable to the step
- The response map library project is eligible to be chained. Response map chaining uses iTest Project Natures to determine whether a project is eligible to be chained. Only projects with the following natures can be chained:
  - **iTest Default Project** nature (For this nature, the icon for the project includes the iTest logo 

)
  - **iTest Response Map Library** nature (For this nature, the icon for the project includes an '**R**' logo 

)

Determining or setting a project's natures

1. In the iTest Explorer, right-click the project and select **Properties**.
2. Click **iTest Project Natures**

Determining which maps are tried and in what order

The Structure view includes a **mappingInfo** section that lists the response maps that are applied during execution.

- Response map library projects that are selected in the primary library's **Project References** page appear in the **projects** section.
- Projects appear in the **projects** list in the order in which they appear on the **Project References** page. If a referenced project itself has referenced projects, then they are added to the list before adding the next project. For example, if A references B and C, and B references D and E, then the list will be ordered A B D E C.
- In the **candidateMaps** list, for each project in order, response maps are sorted in priority order (as specified on the Response Map editor's **Applicability** page)

In this example **mappingInfo** section in the Response view, while the **R5000_router_response_maps** library was selected, it contained no applicable maps and the

S100_switch_response_maps library was found to include two applicable maps. The **show_version** map has higher priority than **show_version2**.

```
<structure>
       <mappingInfo>
              <projects>
                     <project>my_project</project>
                     <project>R5000_router_response_maps</project>
                     <project>S100_switch_response_maps</project>
              </projects>
              <candidateMaps>
                     <map>project://my_project/show_version.ffrm</map>

<map>project://S100_switch_response_maps/show_version.ffrm</map>

<map>project://S100_switch_response_maps/show_version2.ffrm</map>
              </candidateMaps>
       <mapped URI="project://my_project/show_version.ffrm">
              ...
       </mapped>
</structure>
```

Adding a new response map library

1. In the iTest Explorer, use one of the following methods to add a new library:
   - Right-click anywhere in the explorer and then select **New > Response Map Library**.
   - Click **File > New > Other**, and then select **Response Map Library**.
   - Click **New**, select **Other**, and then select **Response Map Library**
2. On the **New Response Map Library Project** dialog box, type the name of the library.
3. Click **Finish**.

Working with response map libraries

Adding a response map to a response map library

You move a response map from a folder into a library by using any of the standard iTest Explorer tools; **drag**, **Ctrl-drag**, **copy-paste**, **cut-paste**, or **move**.

Renaming, deleting, or moving a response map library

In the iTest Explorer, right-click the library and then select **Rename**, **Delete**, or **Move**, as needed.

Specifying that a session profile should use response maps from a particular response map library

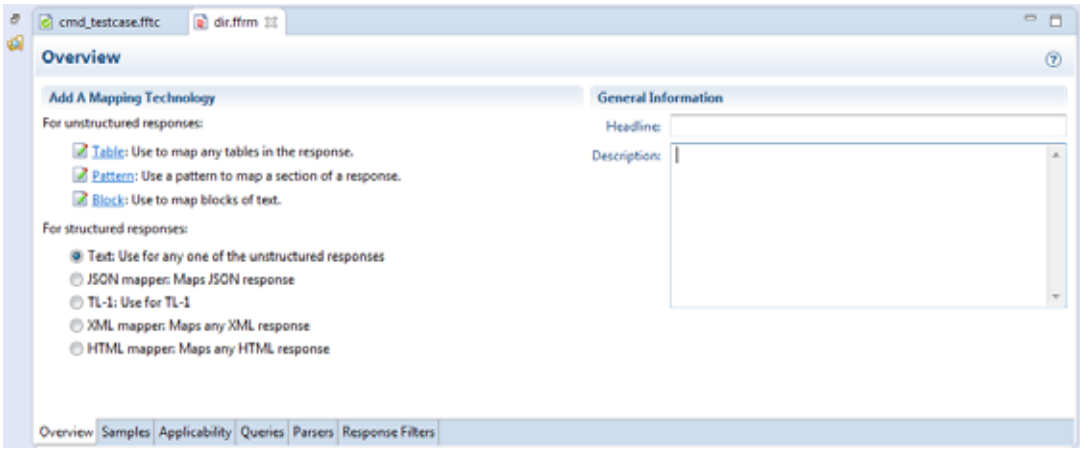
1. By default, each test case step is configured to use the response map library associated with the session profile for its session (**Step Properties** section, **Other Postprocessing > Expected Response**).
2. Open the session profile that should use a response map in the library.
3. Click the **Misc** tab.

4 Specify the library in which to look for response maps that apply to this response map. Be sure to save the session profile.

Tip The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries. See "Making use of existing response map libraries: Chaining response maps" on page 559.

Response Map editor: Overview page

When the Response Map editor opens to the **Overview** page, you have access to a variety of mapping tools.



Tip You can use the **New Response Map** wizard to have iTest take a first try at automatically generating a map.

In many cases, the process generates a map and queries that you can use immediately. In some cases, you may have to rename particular queries. In other cases, the map will not meet your needs and you will have to generate a map using the Response Map editor. See "Creating a response map: Instructions" on page 554.

Add a Mapping Technology

Click a link to open a mapping page for a particular mapping type. (The **help me choose** link can guide you in choosing the mapping technology that is appropriate to the response that you are currently mapping.

Table	Open the **Table Mapping** page. See "Response Map editor: Table Map page" on page 577.
Pattern	Open the **Pattern Mapping** page. See "Response Map editor: Pattern page" on page 574.
Block	Open the **Block Mapping** page. See "Mapping a response using a Block Map" on page 585.

For structured responses

Specify the type of response that you are mapping. All CLI responses other than HTML and XML are unstructured text (the default selection). For HTML and XML responses, the queries are auto-generated, so you can move immediately to the **Queries** page to manage the queries that you expect test case developers to use.

iTest auto-maps JSON, TL1, SNMP, and all traffic generator device responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the generated queries.

General Information

URI	The location of the response map document.
Notes	Type information that describes the response map into the Notes box. For example, you might specify which versions of system software the maps support.

Other tabs/pages

Applicability tab	Open the Applicability page. See "Response Map editor: Applicability page" on page 567.
Queries	Open the Queries page. See "Response Map editor: Queries page" on page 569.
Parsers	Open the Parsers page. See "Mapping JSON responses" on page 600.
Samples	Open the Samples page. See "Response Map editor: Samples page" on page 563.

Response Map editor: Samples page

On the **Samples** page, you provide the full text of both a command and the response (if you created the response map using the **New Response Map** wizard, then iTest populates both the **Response** and the **Command** text boxes). In addition, you specify a name for the sample.

For a sample to be used to generate mappers

You will work with the sample response text while defining the mappers (any combination of Block maps, Pattern maps, and Table maps) that make up the response map.

In some cases, you will configure one sample to represent one form of the response and a different sample to represent an alternative form of the response (details appear later in this topic).

For a sample to be used an emulated response

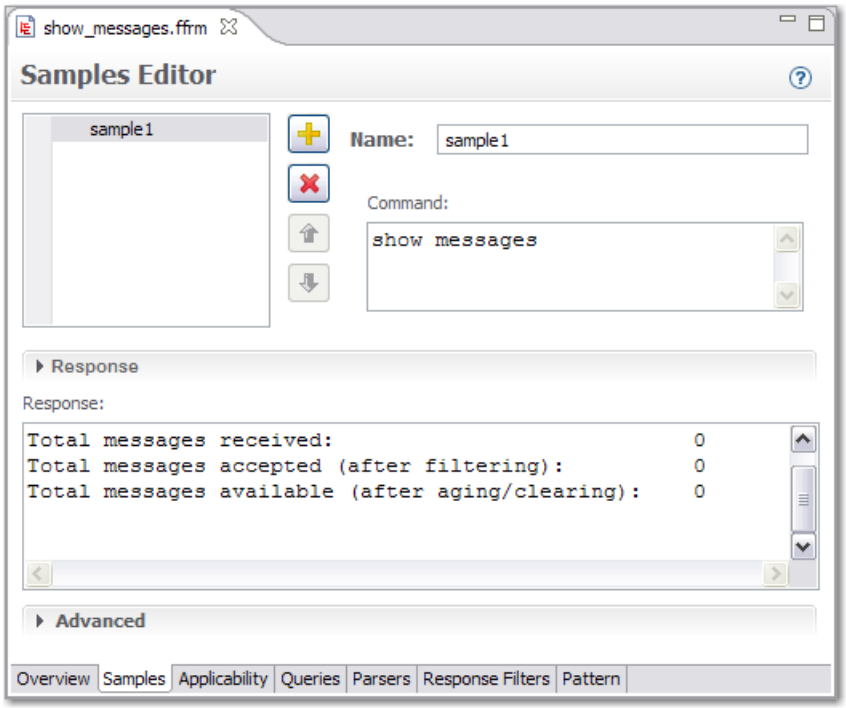
If you plan to use the response map solely to supply an emulated response, then you specify the sample response and do not define any mappers for the response (Block maps, Pattern maps, or Table maps).

Configuring a sample response

While you work on a response sample in the **Samples** page, the Response, Structure, Queries, and Step Issues views provide auto-updated feedback on how the response map will operate.

1 The **New Response Map** wizard (or **Add this Response to an Existing Response Map**) populates the **Command** and **Response** text boxes. (If you are adding a sample response manually [not typical] you paste the values into the appropriate text boxes.)

You can edit the **Response** text as needed, but remember that all mappers that you define map against the text in this box (not an issue if you intend to use the sample as an emulated response).



2 The **Sample name** defaults to `sample1`

Note The **Sample name** is not important if the response always appears in only one format and this is the only sample that you will use for mapping the response. The **Sample name** serves to uniquely identify each of multiple response formats.

The name is important only in either of the following cases:

- You plan to provide more than one sample for the response map. We recommend that you use a name that reflects the particular software revision or other command category that results in the different response. If you're adding a second or third format of a response, you might include text that identifies the format of the response
- You intend to use the response map only to supply an emulated response. The **Sample name** property in the **Emulation** property group for the step will refer to this sample name.

Buttons that manage the list of samples:

Add	Add a new response sample. You can use this button to add the first sample or to provide additional samples for an existing response map.
Remove	Delete the selected response sample. Any maps that you created using the deleted sample are still in place and will be used when mapping a response.
Move Up / Move Down	Use these buttons to organize the samples in the list. **Note** This feature is purely for your convenience and has no bearing on mapping (all response maps are applied to each response regardless of the order of the samples in the list).
Cut	Cut the selected response sample definition (typically for pasting into a different response map document).

3 If you intend to use the sample as an emulated response, then open the **Advanced** section (click the Advanced arrow) and set the following properties. If not, then skip this step.

Do not map this sample	When you check the box, iTest will not generate the mapping logic that enables iTest to return data from responses. • If you edit a sample response so that it will no longer map a response (for example to provide a different emulated response) then you can check the box to prevent a mapping error when the response map is validated. • If you intend to use the sample only as the source for emulated steps, then check the box to avoid mapping errors during execution. Default: unchecked
Content type	Specify the format of the response data. This setting ensures that structured responses like XML and TL1 are correctly parsed. See "Mapping TL1 responses" on page 1216. Default: text

Structured data	This multi-line text box holds the structured part of the response that will be returned for an emulated step. When the sample was added from a response, this text box displays the structured data that was returned in the response. You can edit this text to check how queries will be mapped for different values (in the Query and Structure views) and to simulate different structured data for emulated responses. • If you created the response map using the wizard, iTest uses the structured data from the response that you used when starting the wizard. • If you added the sample by clicking **Add** in the Response Map editor, then this property is blank and you must provide a value. As with the **Response** text, you can modify the text to meet your needs. **Note** During execution, iTest emulates only structured data from the response; it does not emulate structured data that is appended by response mapping.
Duration	The **Duration** value shows how long it took for the step to execute (for the sample that was added from a response). You can use the **Duration** value to simulate the original execution speed when a step is emulated from this sample or modify the setting as needed. To emulate the response with this duration, you must check **Enable emulation duration for the test case** on the **General** page of the Test Case editor. Specify **0.0** to execute as fast as possible.

4 At this point, you have configured a sample response for the response map. Save the response map document by clicking **Save** in the main toolbar.

Adding additional samples to a response map document

You might choose to add more than one sample response to a response map in the following situations:

- The response for a command can appear in more than one mode or format, depending on the state or configuration of the device
- The response to a particular command differs slightly for different versions of the device's software. By designing a single response map that can map the responses from multiple software versions, you do not have to prepare several response maps.
- The sample will provide the emulated response to a step or session. For an overview of emulation, see Chapter 23, "Virtual Testbeds (VTB): Testing with Emulated Sessions".

◆ Recommended method: Click 'Add This Sample to an Existing Response Map'

1 On the Response view, click **Add This Sample to an Existing Response Map** .

2 Once you specify the response map in a dialog box, iTest adds the new sample and opens the response map document in the **Samples** page of the Response Map editor.

3 In the **Sample name** field, type a name. We recommend that you use a name that reflects the particular software revision or other command category that results in the different

response. If you're adding a second or third format of a response, you might include text that identifies the format of the response.

4 Save the response map document by clicking **Save** in the main toolbar.

- **Not Typical: Add the sample manually**

1 On the **Samples** page, click **Add** to add a new sample definition.

2 In the **Sample name** field, type a name. We recommend that you use a name that reflects the particular software revision or other command category that results in the different response. If you're adding a second or third format of a response, you might include text that identifies the format of the response.

3 Drop or paste the appropriate text into the **Command** and **Response** text boxes.

4 Save the response map document by clicking **Save** in the main toolbar.

Deleting a sample

1 On the **Samples** page, select the sample in the list.

2 Click **Remove** to delete the sample definition from the list.

Changing the order of samples in the list

You can use the **Up** and **Down** to organize the samples.

Note This feature is purely for your convenience and has no bearing on mapping (all response maps are applied to each response regardless of the order of the samples in the list).

Response Map editor: Applicability page

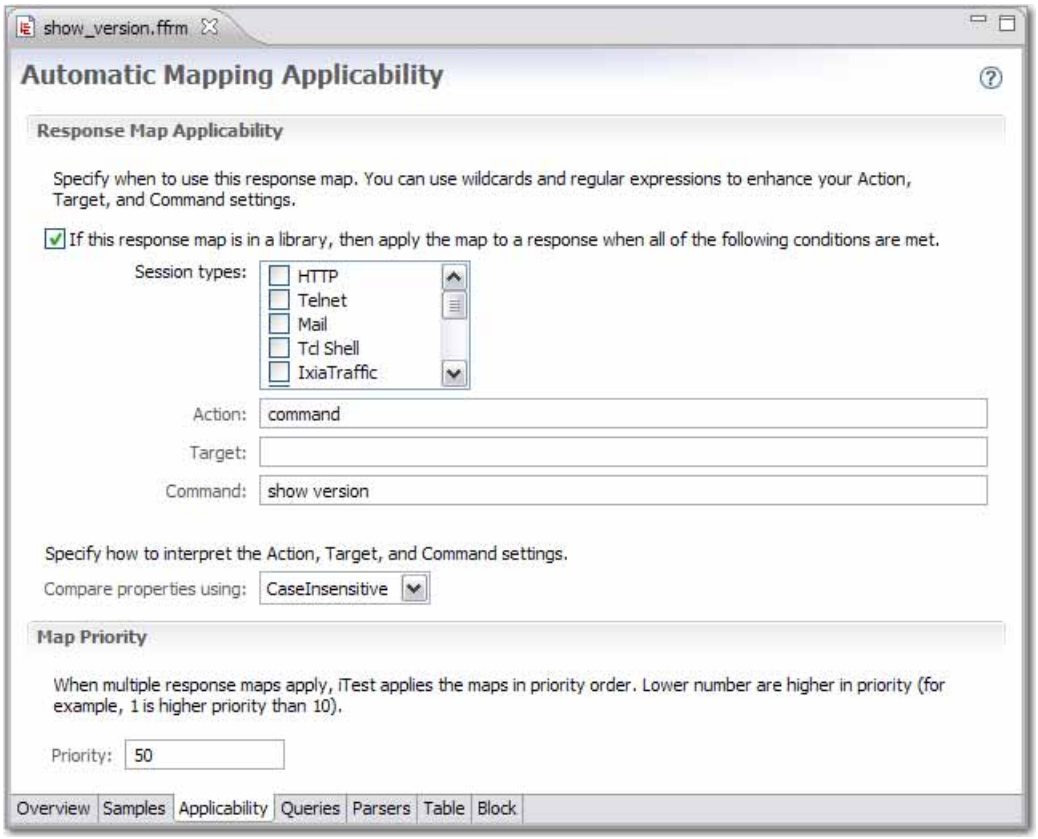
Use the **Applicability** page to specify when to use the current response map to search for matches in a response. For example, you can specify that the response map should be applied to the response whenever a step in a Telnet or SSH session uses a command that includes the text show interfaces*. As a result, iTest applies the response map whenever any of the following commands is executed in a Telnet or SSH session: show interfaces all, show interfaces 2, or show interfaces 2-7.

Here is why you might want to take advantage of this powerful feature: Once you have set the applicability properties and associated the response map with the session profile used in the test cases, you no longer need to explicitly specify this response map for each test case step that should use it. Instead, whenever a test case executes and the applicability conditions are met (for example, a **show interfaces all** command is issued in an SSH session), then iTest applies this response map (and possibly other applicable maps, in priority order) to the response to extract the data for post-process analysis.

You may also filter the applicable response map list as follows. Add one or more options and filter the list of applicable response maps.

Session Types	Select one or more session types to filter on session type
Action	Each session type supports different actions: Use this field to filter on action, e.g, snapshot.

Target	Some sessions support targets (e.g. Selenium. See “Selenium sessions” on page 1652.), this field can be used to filter on a target.
Command	This is the most commonly used filter. Use to filter on a particular command. **Note** You may use wildcard characters. For example, a command filter `sh* ver*` will filter applicable commands: `show version` and `sh version`



CAUTION If you do not specify applicability settings for a response map, or move the response map out of a response map library (not recommended), then the map will be used only if it is explicitly specified in the test case.

Setting applicability properties for a response map

Whenever a test case step executes, iTest compares the session type, action, command, and other conditions of the step to the applicability conditions specified for all response maps that appear in response map libraries associated with the session profiles for sessions in the test case. For each map, if all of the conditions match, then iTest applies the map to the response in priority order.

1 To set the applicability properties, first open the **Applicability** page and then configure the following property settings:

Session types	Optional: Select all session types for which this response map applies.
Action	Optional Specify the action in a step that results in a response to which the response map should be applied.

Command	Specify the text string of the command in a step that results in a response to which the response map should be applied.
Compare properties using	Specify how to interpret the contents of the properties.
Map Priority	When multiple response maps apply, iTest applies the maps in priority order. Lower number are higher in priority (for example, 1 is higher priority than 10). Specify the value so that users can choose the order in which multiple applicable maps are considered.

2 Save the response map.

3 Add the response map to a response map library. The map must appear in a library because sessions and steps check configured to look for response maps will test whether this response map is applicable during test case execution.

Note The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries.

Response Map editor: Queries page

The **Queries** page allows you to add and edit custom query definitions. Use custom query to define a friendly name for an XPath query and optionally, to add arguments to the query.

Custom queries enable you to define your own XPath functions to provide a friendlier name for a complex XPath query or to parameterize a query with arguments. A query consists of a ***query format string*** (in XPath syntax) with possible substitution markers in the form of **{0} {1} {2}**, and so on.

You must use a response map to make use of a custom query.

Note If you have defined all of the custom queries that automation engineers will need while creating test cases, then you can check **Never show automatically generated queries.** As a result, only custom queries will appear in the **query** drop down lists in the Test Case editor.

Example 1

rowCount is a friendly name for the query format string **fn:count(//table/row)**

So, in a test case, **rowCount()** counts the number of rows in a table

Example 2

Let us define a query to find the interface status of a row with interface name **Eth0**

Let us name the query **ifStatus**

We will pass in one argument: **ifName**, so, let us define an argument named **ifName**

For most arguments, you need only supply a default value.

The query format string would be **//table/row[ifName = '{0}']/ifStatus**

Defining a custom query

The easiest way to define a custom query is to modify one of the auto-generated queries (a query generated by the iTest mapper).

You can use the Structure view to test and develop a query. Click **Evaluate** in the Structure view to view how well the query performs (what it matches). Update as needed and evaluate again. When you have what you need, add it to the list of queries for a response map by pasting it into the **Queries** page in the Response Map editor.

You can set preferences for mapping queries. See "Setting preferences for response mapping" on page 601.

Auto-generated queries

iTest displays the list of queries (and the equivalent XPath format) that the mapper generated.

See "Mapping JSON responses" on page 600.

See "Mapping TL1 responses" on page 1216.

Creating a custom query

1. Select the query in the list. Click **Make Custom**
2. The query is added to the Custom Queries section: The name appears in the **Queries** list, the name is added to the **Query name** box, and the XPath is added to the **Query format** box.
3. Now edit the values as needed:

Custom queries

Query name	Specify a name for the custom query.
Query format	Specify an XPath expression with optional argument substitution using zero-based argument numbers in the format **{0} {1} {2}** For example, assume that there is one argument that is the **ifIndex**: **//row[ifIndex='{0}']/ifDesc**
Description	Type text that will help coworkers understand the use of the query.

4. If appropriate, click **Add**  to add arguments and specify property settings as needed:

Arguments

Argument name	Specify a name for the argument.

Optional advanced properties

Default value	Optional. Type a default value. Later when a test case that uses the query is loaded for execution or paused, you can change the value in the Data view.
Values query	Optional. Provide an XPath query that finds all possible values for the argument within the structure data. iTest shows all possible cases in the Queries view. This means that users can right-click the blue boxes in the Response view to add a query-based analysis rule. (Most auto-generated queries make use of this technology). You can try the query out by pasting it into the Structure view and clicking **Evaluate**. When the query returns the values you expect, copy/paste it back into the **Values query** property box.
Interpret as	Optional. Specify how to interpret the **Values query** setting. **DontInterpret**: Ignore the **Values query** setting. There will be one value for the argument supplied by the **Default Value** property. **SampleValues**: Values for the argument will be the *values* of the nodes returned by the **Values query**. **ItemCount**: When the **Values query** returns a list, then you can specify **ItemCount** to so that the values for the argument will be 1, 2, 3, … up to count of items returned by the query. For example, use this setting to get the count of rows in a table. **SampleNodeNames**: Values for the argument will be the *names* of the nodes returned by the **Values query**.

About XPath functions

For an XPath language description, see

http://www.w3.org/TR/1999/REC-xpath-19991116

iTest supports the following XPath functions:

http://jaxen.org/apidocs/index.html

Supported XPath functions

iTest supports the following XPath functions:

http://jaxen.org/apidocs/index.html

To support backward compatibility with FanfareSVT, iTest supports the following additional XPath functions:

min

max

avg

For an XPath language description, see:

`http://www.w3.org/TR/1999/REC-xpath-19991116`

Response Map editor: Parsers page

The process of response mapping involves querying the structured part of the response and returning the result of the query. iTest uses built-in parsers to identify the structure. For example the date parser identifies strings of the form **03-16-2011**.

Some organizations have developed parsers for responses that reliably return the data required for the test. The **Parsers** page allows you to add and edit your organization's custom regex parser definitions.

Important For Block maps: When you add, delete, or make any change to a custom parser, follow the procedure to reparse the blocks as described in Step 5:

Adding a custom parser

1. On the Response Map editor, open the **Parsers** page.
2. Add the parser: Click  and specify a **Name** for the parser, for example, **IP_Hex**. Use a common-sense name that will help your coworkers to understand its usage. The name must be unique among built-in parsers and other custom parsers.
3. Specify values for the parser properties. You must specify a value for the **Regex string** property. All other properties are optional.

Group name	Optional. The name of a group (defined within the RegEx String) that identifies the portion of the match that is to be treated as the token. If blank, then the entire match (if any) with the Regex string is the token.
Regex string	Required. Specify a regular expression that defines the structure and content of the single token that the parser is intended to find. For example, the following RegEx String matches IP addresses in hex form in dotted or not dotted notation (not tested): [0-9A-Fa-f]{2}(\.?)[0-9A-Fa-f]{2}(\.?)[0-9A-Fa-f]{2}(\.?)[0-9A-Fa-f]{2}
Precondition Pattern	Optional. If there is any text before the parsed token, then it must match the character defined by this RegEx.
Postcondition pattern	Optional. If there is any text after the parsed token, then it must match the character defined by this RegEx.
Priority	Optional. The priority ordering in which the parser is applied to the response. Negative values cause the parser to have the higher priority. Non-negative values define a specific priority for the parser. The higher the number, the lower the priority. See the next table for priority settings of iTest default parsers. Default: 0
Default is variable	Optional. True: Tokens found using this parser are considered to be variable. So, when mapping, the parser accepts a different token value of the same type in this position. False: The parser requires that the token in this position must have the same value as the original.

4. Save the response map. Close and reopen the Response Map editor.

5 For Block Maps: Open the **Block** page. Reparse all the blocks that include the field that you created the custom parser for: In the **Block Map Elements** list, select the blocks (Ctr+click to select multiple) and then click **Reparse Blocks**.

A dialog box warns you that reparsing could cause you to lose token names and other information. Click **OK**.

Buttons on the Custom Parsers page

Add	Add a new custom parser. **Note** When you add a custom parser, you must then reparse all the blocks that include the field that you created the custom parser for: In the **Block Map Elements** list, select the blocks (Ctr+click to select multiple) and then click **Reparse Blocks**.
Remove	Delete the selected parser. **Note** When you delete a custom parser, you must then reparse all the blocks that include the field that you created the custom parser for: In the **Block Map Elements** list, select the blocks (Ctr+click to select multiple) and then click **Reparse Blocks**.
Move Up / **Move Down**	Use these buttons to organize the parsers into the order in which they should be applied.

Centralizing custom parsers

You can centralize your custom parsers in a single common map in a response map library.

Create a response map called **_common.ffrm** and put it in the root directory of the library. All of the custom parsers from the map will be used when performing any mapping that uses the library.

Built-in iTest parsers

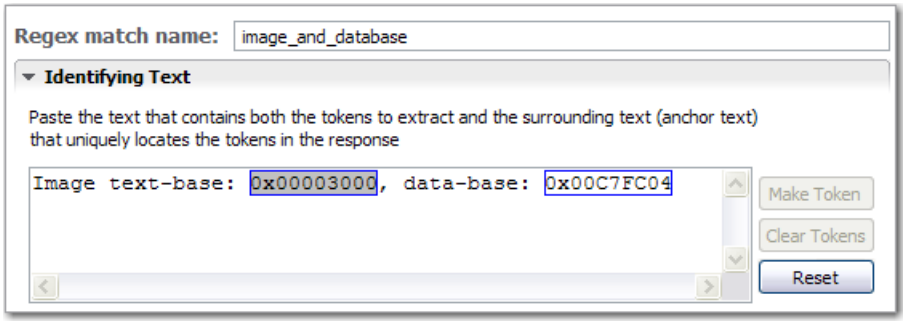
Parser Name	Priority
Control	100
Punctuation	200
Whitespace	300
Word	500
Number	700
HexNumber	1000
Timespan	1100
TimespanDecimal	1200
IPv4Address	1300
Uptime	1400
TimestampNumericDate	1500
MacAddress	1600
LinuxUptime	1700
OtherTimestamp	1800
TimestampTimeDate	1900
HttpTimestamp	2000
IPv6Address	2200
LongAlphanumeric	2300
Base64	2400
LongSpaceSeparatedHexString	2500

Response Map editor: Pattern page

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

You define a Pattern map to match (and therefore extract) data from an identifiable text string in a response. Often, in a single text string from the sample response text, you will identify

several regular expression groups (tokens) that will eventually extract data from real responses during test case execution.

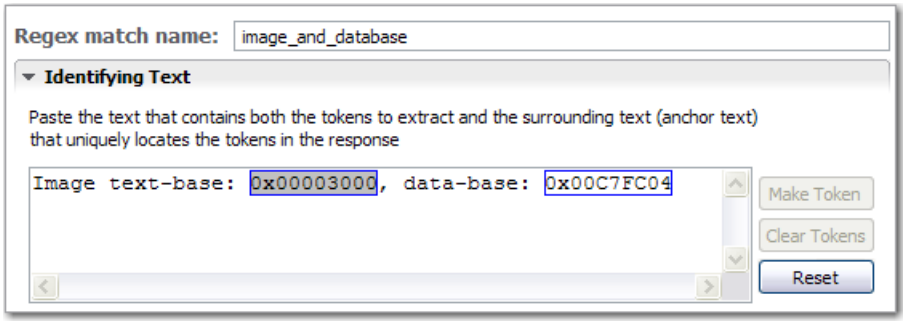


Defining a Pattern map

1. Click **Add** to add a new pattern match definition.
2. In the **Name** box, type a name that represents the values that the pattern will extract. For example, you might name a pattern **image_and_database** because you can extract both the image text-base and the database from the line (see the example in Step 4).
3. In the Response view, select a fragment of the response that contains only enough text to define the context for the information that you want to extract. Sometimes, to ensure that the text is the unique way to find the value, you will need to include text from the line before or after, or even several lines. There are two options for providing the pattern that includes the matches:
4. In the sidebar, click **Add Pattern**. iTest pastes the selected lines into the **Identifying Text** box.

   Alternatively, Copy the text and paste the text into the **Identifying Text** box.

   iTest immediately attempts to identify values that you want to extract. iTest draws a blue box around each group of interest (numbers, timestamps, IP addresses, and other types of response value that you might typically want to analyze). If you modify the text, iTest immediately updates the groups.(To disable auto-update, uncheck **Automatically update definitions to maintain consistency with the text in the Identifying Text box**.)



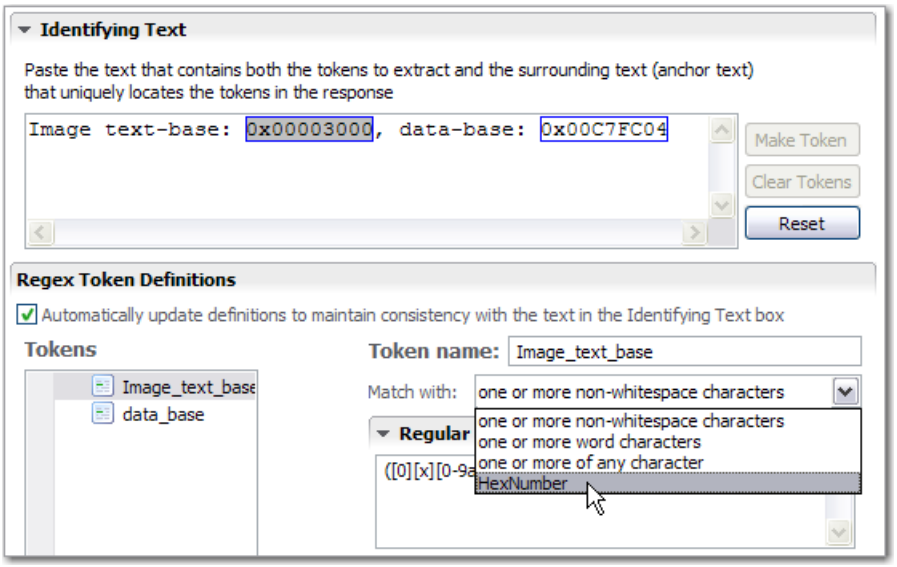
   At this point, you have identified one or more ***tokens*** (in blue boxes) and queries that extract the token values. The groups of text between the tokens are anchors. Anchors function to locate the tokens whose values you want to extract. By default, iTest creates default names for tokens by using the anchor text. In the example, iTest names the 0x00003000 token **Image_text_base** and the 0x00C7FC04 token **data_base**. Depending on the order of the text, iTest might use the text that occurs after a token to form its “best guess” name.

5 If iTest does not specify a token correctly or misses a value that you want to extract, then select the actual value that you want to extract and click **Make Token**. Other controls:

Clear Tokens	If iTest has identified a token that you are not interested in: Click anywhere in the token text. Click **Clear Tokens** to change the text into an anchor group.
Reset	Click **Reset** to revert the Token definitions to the initial settings suggested by iTest. **Reset** is useful when you have made changes to token definitions that you do not want to save,

6 Optional. Check **Generate an error if no matches are found** to specify that an error should be generated when the token does not appear in the response. Errors appear in the Execution view, in the Step Issues view, and in test reports.

7 In the **Identifying Text** box, select a named token. In the **Match with** box, iTest suggests a regular expression that matches the selection and could extract the value. The associated actual regular expression appears in the **Regular Expression** box. From the **Match with** box, select the regular expression that best matches the field. In our example, we realize that the values will always be hex numbers, so we can safely select **HexNumber.**



8 You can modify the regex in the **Expression** box as needed.

Other controls:

Extract as structured data

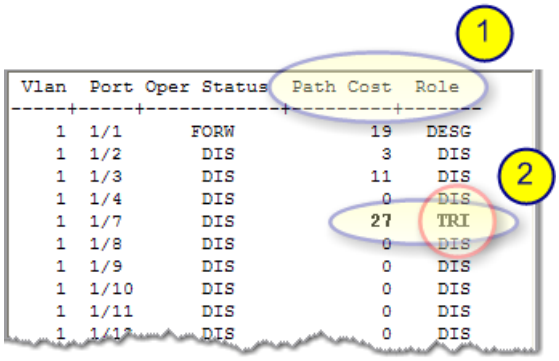
Advanced Users only. Uncheck this box to convert an auto-generated pattern to an anchor.

Use this group as a key when generating aliases

(Optional) In some cases, you will use a particular value for a token as a key — that value identifies the line in the response where you want to extract another value from the line. In the example the regex map finds each row in the table and the groups extract the cell values in the row (discarding the whitespace or other delimiters). In this example, we defined **PathCost** as a key token:

1 The PathCost token is the Key. Whenever a PathCost value exceeds 25, then the RoleByPathCost query returns the value in that row for the Role token.

2 The PathCost token in this row exceeds 25, so the RoleByPathCost query returns the value in this row for the Role token: TRI.



Check **Use this group as a key** and provide a **Sample key value**. iTest uses the Key token to auto-generate aliases for the other tokens in the block.

As you make changes, the Response view updates itself to reflect the changes. The text in the Response view is linked to the selected pattern match. Each match in the response body to the overall pattern map is highlighted with a blue background. Each extracted group is enclosed in a blue box. If one or more of the groups is marked as a key, that is shown with a different font color.

Describe How to Search

These are optional settings.

Line-by-line	Search for matches one line at a time.
Raw	Search for matches over the full response text.

Response Map editor: Table Map page

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

The Table Map technology works in conjunction with visual cues in the Response view — lines that mark column boundaries, color coding, and so on.

Why use Table maps?

The goal for mapping a table is: For a particular response that includes data in a table format (even if there are other tables or other non-table data in the response), to be able to extract a token value from any cell in a table (some cells might even contain more than one token value).

- You'll use the Table Map editor for each type of table to specify:
- The method for identifying the beginning and end of the data in the table
- The method for delimiting columns: for example, by tab characters or by fixed column widths
- Whether the table can expand/contract with additional columns to cover additional data
- The minimum and maximum number of instances of this type of table to expect in a response
- Once you have defined the properties for the table, you will specify properties for each column in the table:

- The width of the column (for the case that the table boundaries are based on column width)
- A default value for the data in the cell
- Methods for dealing with non-standard, too wide, or blank contents

Tip You can set preferences for table mapping. See "Setting preferences for response mapping" on page 601.

Defining Table maps

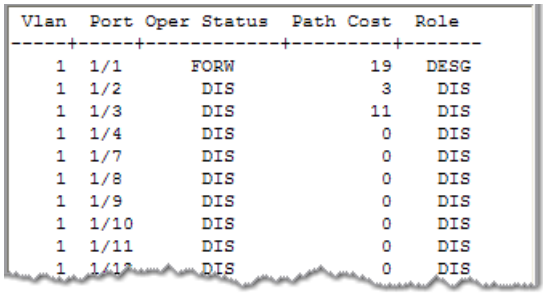
This topic describes how to specify the table structure using the Table Map editor pages. As you gain confidence, you can use the Response view sidebar (click in the Response view) to quickly design table maps.

General Properties

Map name	Type a name. We recommend that you use a name that reflects the particular device configuration or other command category that results in the particular table.
Description	Optional. Provide text that explains how the table map is structured or used.
Remove rows with empty key fields	If the table includes rows with empty key fields, then select this option to remove them.

Table Location: Start

Tables can appear anywhere within response text (other response text or other tables might appear before the table that you're defining). The **Start** property settings enable you to specify how to find the start of the data in the table. We'll refer to this example table:



```
Vlan  Port Oper Status  Path Cost  Role
-----+-----+------------+---------+-------
   1  1/1       FORW            19   DESG
   1  1/2        DIS             3    DIS
   1  1/3        DIS            11    DIS
   1  1/4        DIS             0    DIS
   1  1/7        DIS             0    DIS
   1  1/8        DIS             0    DIS
   1  1/9        DIS             0    DIS
   1  1/10       DIS             0    DIS
   1  1/11       DIS             0    DIS
   1  1/12       DIS             0    DIS
```

You can specify either of the following methods for locating the start of a table:

The first row of data appears after a banner

If the table always starts with a recognizable line of text (typically the title of the table or a row of column headings), then you can specify this *banner* text as the unique identifier for the

beginning of the table. Our example table includes a unique line (the headings **`Vlan Port`**, and so on) followed by an additional line (the **`---+---`** characters)

Banner contains	Type or paste all or part of the banner text into this box. In our example, we would paste the following text: Vlan Port Oper Status Path Cost Role
Match banner using	**CaseInsensitive**: Match the footer text without regard to type case. **CaseSensitive**: Match the footer text only if all type case matches also. **Regex**: Match potential footer text against the regular expression that appears in the Banner contains text box. **Wildcard**: The * character in the Banner contains value represents any number of characters. The ? character in the Banner contains value represents a single character.
Number of additional lines in banner	Specify the number of lines that appear after the banner line. In our example, one additional line appears after the banner (the row of ---+--- characters), so we enter **1**.

The first row of data starts on a specific line in the response

A table might always appear at a particular location in the response (for example, it might be the first item in the response or might always appear on the eighth line). In this case, you can specify the **Line number** at which the data in the table starts (starting with line number 0). The first line in the response is 0, the second line is 1, the third line is 2, and so on. If our example table were the first item in the response, we would tell iTest to skip the first two lines in the response by specifying a **Line number** of **1**.

Table Location: End

Many responses include text after the table, so you can use the **End** property settings to specify how to determine when the table data ends. You can specify that the data ends either:

- On the row before the first blank line: The data ends either when the response text ends or when the mapper encounters a blank line.
- At the end of response: The table data begins after the response. When you seelct this option, you may choose to **Ignore blank lines found within the table.**
- On the row before a footer: The data ends when unique footer text appears. (The mapper does not attempt to map the footer.) If you specify a footer, then you must specify the following properties:

Footer contains	Type or paste all or part of the text of the footer.
Match footer using	CaseInsensitive: Match the footer text without regard to type case. CaseSensitive: Match the footer text only if all type case matches also. Regex: Match potential footer text against the regular expression that appears in the Footer contains text box. Wildcard: The * character in the Banner contains value represents any number of characters. The ? character in the Banner contains value represents a single character.

- Ignore blank lines found within the table: The option is available when you select, **At the end of response.** If the table includes blank lines (for example, at otherwise meaningless page breaks), then select this option.

Note If you will identify the end of the table as the first blank line, then leave this box checked.

Table Location: Repeating Tables

Use this page to specify how to respond when multiple instances of the table appear within a single response.

Required	Generate an error if the table is not found at least once. The error appears in the Execution Messages window and in test reports whenever the table does not appear in a response.
Multiple	Check this property if the table can repeat within a response. If the table appears more than once in a response, then use the data in all table instances. **Note** This setting is allowed only if the start of data in the table is found using a banner (as described in "The first row of data appears after a banner" on page 578 and also later in this topic).

Locating Columns

On this page, you specify how to recognize a new column of data in the table. We'll refer to this example table:

```
Vlan  Port Oper Status  Path Cost  Role
-----+-----+------------+---------+-------
   1  1/1       FORW            19   DESG
   1  1/2        DIS             3    DIS
   1  1/3        DIS            11    DIS
   1  1/4        DIS             0    DIS
   1  1/7        DIS             0    DIS
   1  1/8        DIS             0    DIS
   1  1/9        DIS             0    DIS
   1  1/10       DIS             0    DIS
   1  1/11       DIS             0    DIS
   1  1/12       DIS             0    DIS
```

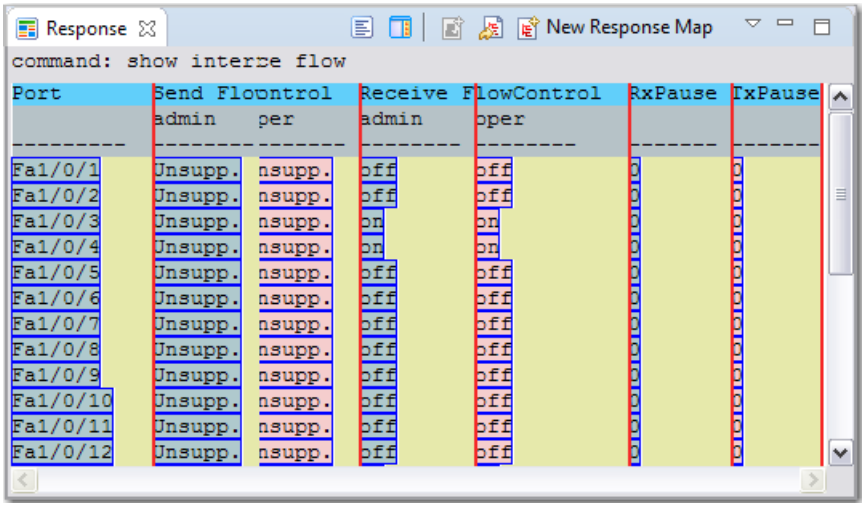
There are two methods for determining column boundaries:

- Delimited: Use whitespace characters (tabs and/or spaces) or some other specified character (for example, commas) to delimit column boundaries

Whitespace	(default) A new column starts upon encountering a space or tab character. Whitespace is the most flexible setting, as it accepts tabs or spaces for separating the data entries. Our example table seems to use whitespace.
Tab	A new column starts upon encountering a tab character only. Note: If you specify Tab and the table actually uses a mixture of tabs and spaces as delimiters, then the mapper will map any space characters that appear between columns as a part of the token data — probably not what you want.
Comma/Colon	A new column starts upon encountering a comma/colon character only.
Custom	If you specify Custom, then specify the delimiter string in the **Other delimiter** text box.
Regex	If you specify Regex, then specify regular expression delimiter string in the **Other delimiter** text box.

- Positional: Apply strict column widths based on character counts (for example, the first column is 8 characters wide, the second column is 14 characters wide, and so on).

While you work on a Table response map that uses character counts to determine column boundaries (the **Positional** setting), the Response view displays column markers that indicate the end of each column of data. To change the location of a column boundary, drag the marker to place it after the last character in the column, as shown in the example:



For our example table, it seems that we could use either method. Follow these suggestions:

Tips The **Delimited** setting using whitespace is typically the easiest setting to apply, as it accepts tabs or spaces for separating the data entries. The example table can be mapped using whitespace.
If you expect blank cells in the table, then you should use strict column widths (**Positional**) to ensure that the mapper does not miss a column boundary. (Because the actual content of a blank cell is whitespace, if you had selected to **Delimited**, the mapper would incorrectly interpret the space as a column boundary).
The **Positional** setting is also useful when you expect space characters within a singe value in a cell.

Preventing column wrapping in responses

In a CLI terminal session, the output wraps at the terminal width. This may cause table output to wrap. To avoid the problem, increase the terminal width, as follows:

1. Edit the session profile.
2. Click **More** to view the properties page.
3. Click **Terminal > Size**.
4. Increase the **Terminal Width** setting (to, for example, **120**) and select either

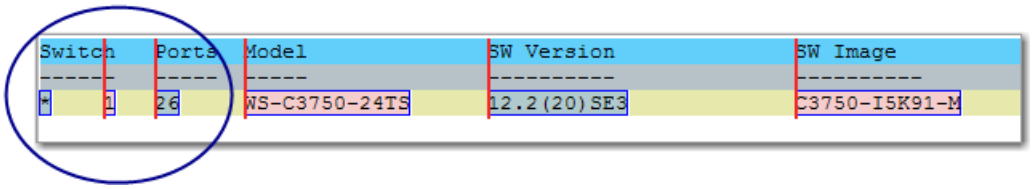
   same as window size with the specified minimum

   or

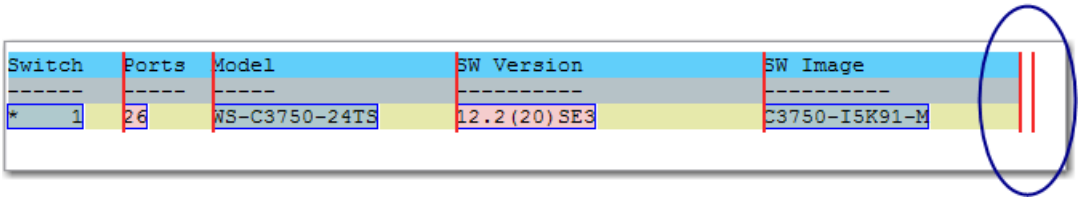
   fixed as specified

Troubleshooting column widths

Sometimes, auto-adding columns mistakes a column with two values in a cell as two separate columns, as in the example.



To fix the map, drag the right-most column border to the far right and then move each border to the next column to the right. Notice that there is now an extra column at the end.



Select the column and delete it.

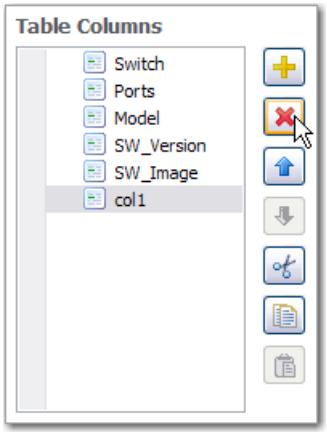
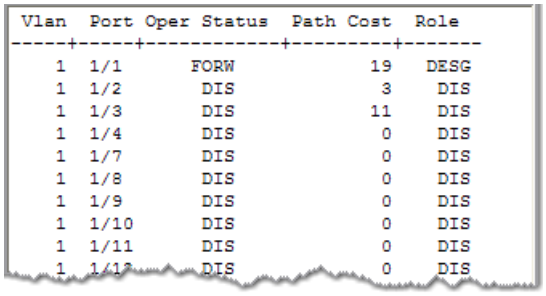


Table Columns

Column name

Type or paste a name for the column. Typically, you name columns with the heading names. For this example table, the column names would be **Vlan**, **Port**, **Oper_Status**, **Path_Cost**, and **Role**.



Vlan	Port	Oper Status	Path Cost	Role
1	1/1	FORW	19	DESG
1	1/2	DIS	3	DIS
1	1/3	DIS	11	DIS
1	1/4	DIS	0	DIS
1	1/7	DIS	0	DIS
1	1/8	DIS	0	DIS
1	1/9	DIS	0	DIS
1	1/10	DIS	0	DIS
1	1/11	DIS	0	DIS
1	1/12	DIS	0	DIS

Key

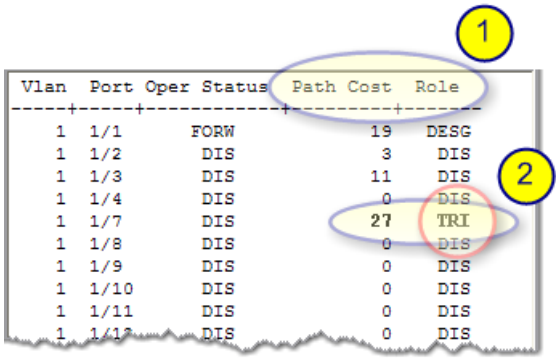
When you check the **Key** option and provide a **Sample Key Value**, iTest uses the value in the column to find particular rows in the table.

Check the Key checkbox and provide a Sample Key Value to use the value of the token as the key to identify the instance of a repeating block in the response that contains the value that you want to extract. iTest uses the Key token to auto-generate aliases for the other tokens in the block.

In this example, we defined **PathCost** as a key token:

1. The PathCost token is the Key. Whenever a PathCost value exceeds 25, then the RoleByPathCost query returns the value in that row for the Role token.

2 The PathCost token in this row exceeds 25, so the RoleByPathCost query returns the value in this row for the Role token: TRI.



Key	Type or paste a sample value that is used to identify the token of interest.
Include the column in the Structured data	Available only when **Key** is not selected.
Create a query for extracting all values appearing in this column	Select to automatically create a query for extracting all values appearing in this column.
Parse cell contents	Divide contents into separate tokens based on parsing rules. This setting enables you to extract multiple values from a single cell if the value cannot otherwise be recognized.
If cell is missing or empty	For empty cells, you can either insert a default values or send an error message to the Execution view and test report. • If you select **UseDefaultValue**, then specify a **Default** value. • If you select **Error** option, only the **Step issues** view and **Error log** view are affected and indicates an error (the **Query view** displays a blank).
If cell has this value	Use this option to replace certain values with the value specified in Translate to.
When cell contents spill over	(Applies only if you have specified Positional — strict column widths based on character counts — to determine column boundaries. Steal: The right column uses the spillover data from the left column. Extend: The left column retains all data in the cell.
Auto populate empty fields	Select **Auto-populate empty fields**, empty cell will be populated with non empty values from the previous row (that is, from the row above the current row).

Custom queries	Allows you to create several simple keys (one field) and compound keys (from several fields).
	For example:
	If your table column has 4 keys: VLAN, IP_Address, MAC, flags
	You may create two keys as follows:
	Simple key with one filed: MAC
	Compound key with two fields: VLAN and IP_Address
	That is, to create queries as follows.
	`//row[VLAN='{0}' and IP_Address='{1}']/MAC`
	`//row[VLAN='{0}' and IP_Address='{1}']/flags`
	`//row[MAC='{0}']/VLAN`
	`//row[MAC='{0}']/IP_Address`
	`//row[MAC='{0}']/flags`
	Create Query List:
	• Click **Add**. The label **Indefinite Key** appears in the **Query** column
	• Click **Indefinite Key** and the **Select key** column lists the current keys (column names).
	• Click on the required key in the **Select key** column and it appears in the **Query** column.
	• Click on another key in the **Select key** column to create a compound/combined key
	Note You may add multiple keys, delete, or move the keys up/down as required.

Use the last column as a template for additional identical columns	Some tables have an indefinite number of columns and expand to the right as needed. For example, the response to a GetStats command can have as many columns as the number of ports in a multi-card device. To allow for this possibility, check The last column may repeat.

Mapping a response using a Block Map

A *block* is an identifiable section of the response that consists of one or more lines of text. You can use block response maps to represent any response that does not have a structured format (indented text is not a structured format).

Create a block response map when the following conditions are all true:

- You want to extract many values from a response (that is, you'd have to create a lot of regex maps to extract all the data you need)
- You want to be able to reuse the map in a variety of test cases
- The data is not in the form of a table. In simple terms, use a block map when the response looks like a paragraph of text with values embedded in the text.

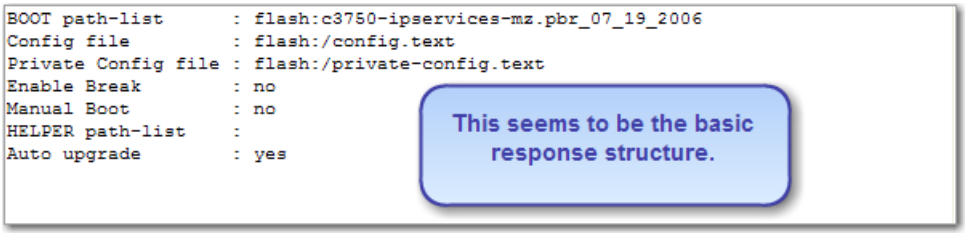
Example block format responses

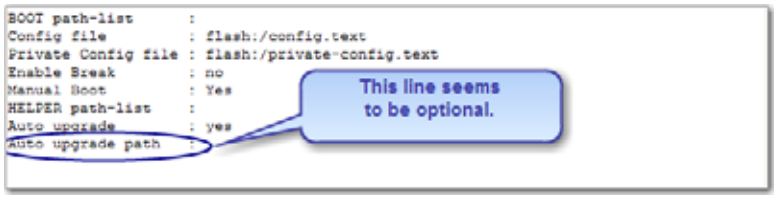
The *non-repeating block* pattern is the most common response pattern. It has the following properties:

- The response consists of a single block of text.
- The identified block of text does not repeat in the response text.
- Typically, but not always, some parts of the text do not change, and some parts vary.
- In the following three sample responses for the same command on a device in different states, you can see that:

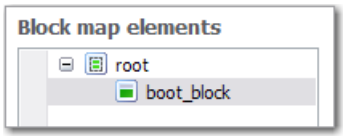
- This block of data appears only one time in the response
- Some lines seem to be optional (for example, **`Auto upgrade path`** appears only in Sample 3)
- The heading portion of each line (the text before the : character) remains unchanged
- The values for each line (the text after the : character) can vary

  Let's look at an example:





Here is the structure of a non-repeating block pattern as represented in the **Block** mapper page:



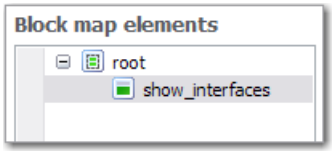
Repeating block

Here is an example of block of data that repeats in the response. You can see that the response is structured as multiple instances of a single block of text. As with a non-repeating block pattern, headings for each line (within a particular block) remain unchanged and the values for each line can vary.

This block of text repeats.

```
Interface Fa1/0/1
---
Port enable administrative configuration setting: Disabled
Port enable operational state: Disabled
Current bidirectional state: Unknown

Interface Fa1/0/2
---
Port enable administrative configuration setting: Disabled
Port enable operational state: Disabled
Current bidirectional state: Unknown

Interface Fa1/0/3
---
Port enable administrative configuration setting: Disabled
Port enable operational state: Disabled
Current bidirectional state: Unknown

Interface Fa1/0/4
---
Port enable administrative configuration setting: Disabled
Port enable operational state: Disabled
Current bidirectional state: Unknown
```

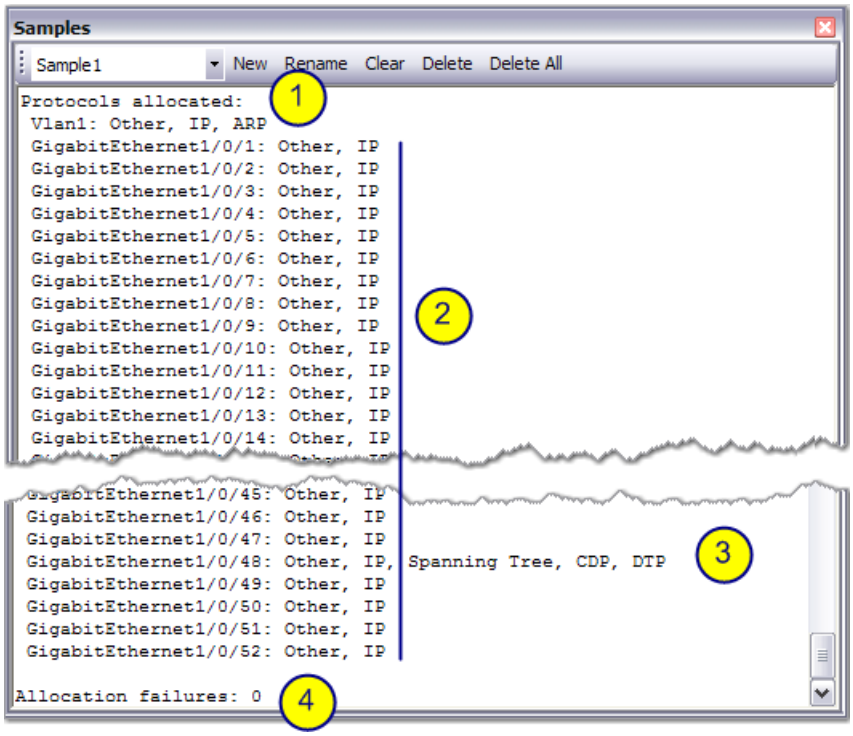
Because you can represent the multiple appearances of the block by specifying that the block can appear more than once in a response, the process of defining the response map is the same as for the non-repeating block pattern.



Partially repeating, multiple-block

The general structure includes a header (optional), a middle section of text with a fixed structure, and (optionally) another header element (you might think of it as a footer).

1 Block: header

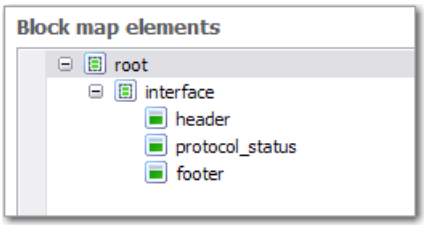


2 Block: protocol_status

3 In this example, this data is ignored. If we needed the data, we could create a block for it.

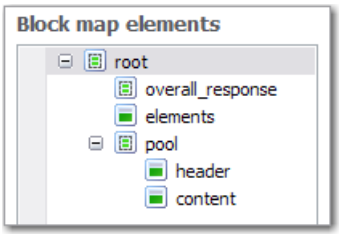
4 Block: footer

Because there are multiple blocks, you will use a container to hold the blocks.



Repeating, multiple-block with containers within containers

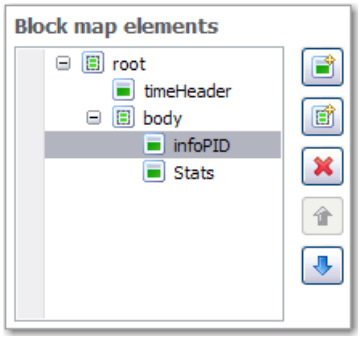
The general structure of this example is just like repeating multiple block with the addition of a container that itself includes a similar structure.



Overview: Properties of Block map elements

Root

The **Root** properties specify how iTest should respond to token and blank line mismatches and whether the elements defined in the response map must appear in the listed order.



Block

The **Block** properties enable you to specify:

- That the block itself (that is, the portion of the response represented by the block) can be absent from a response or can appear more than once in a response.
- The expected contents of the block. You Will use the contents that you paste into this box as the basis for line and token definitions (described in a moment).

Container

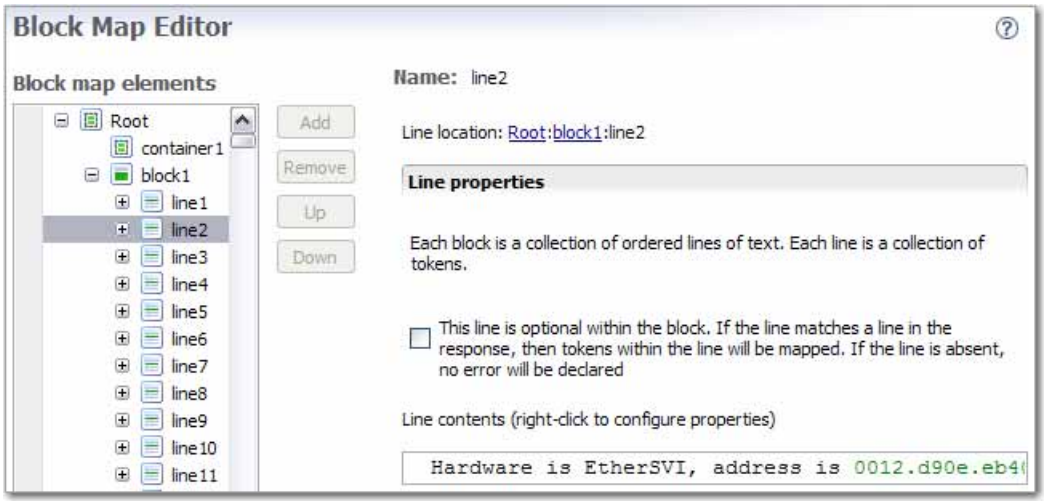
You create a container when a response map should contain more than one block. Unlike blocks, containers do not contain any response text. Containers function only to hold groups of blocks or other containers (subcontainers) that hold other blocks (and possibly further containers). When you select a container, the **Properties** window changes to reflect container-level properties. In the example, the **body** container holds two blocks: **infoPID** and **Stats**.

Container-level properties are the same as block-level properties with a single addition; you specify whether the contents of the container must appear in the listed order or in any order. By default for any container, the listed order of blocks and subcontainers within the container is the order in which they must appear in the response.

Line

When you select a line in the block definition, the **Line** properties appear. The text of the line appears in the **Line contents** box. Within the line, non-text tokens appear in green text and named tokens are enclosed in blue boxes.

You can specify whether the line must appear in the block and, by right-clicking in the text, you can set other properties for the line (for example, that the line can have any contents) or for tokens in the line.



Token

A *token* is the text returned by a query. Queries extract the particular value of interest when the response map that you're working on is applied to a response. You identify a token by selecting and right-clicking the text in the **Line contents** text box and then naming the token and setting other properties.

Response Map editor Block page: Creating a block map

On the **Block** page, you identify tokes in an unstructured response.

Let us map the response to the **`show interfaces`** command. The response includes multiple blocks of data — one block supplies VLan data and another block supplies FastEthernet data. Because there are two distinct block formats in a typical response, we will create two block mappers — one for VLan blocks and one for FastEthernet blocks.

Tip You can set block mapping preferences. See "Setting preferences for response mapping" on page 601.

Important When you add, delete, or make any change to a custom parser for a block map, follow the procedure for reparsing the blocks as described in "Response Map editor: Parsers page" on page 572.

Define the block

First, we define the block that will map (return the required data from) any Vlan block of data in a response.

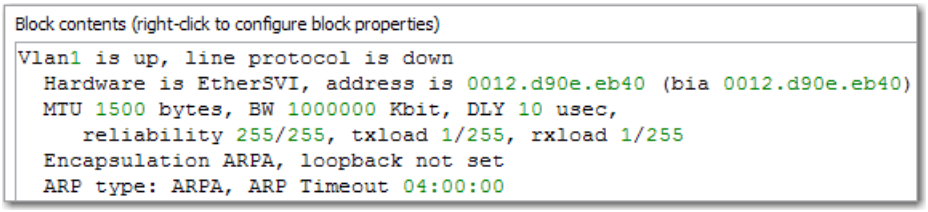
1 In the Block Map editor, click **Add Block** to add a new block to the root node of the response map.

2 In the **Name** box, type a name that represents the purpose of the block. For example, you might name the block **vlan** because it represents a block of data in the response that lists the configuration of a particular VLan interface.

3 In the Response view, select and copy one representative block of text. If the blocks are separated by blank lines, then include the blank line that appears after the text. In our example, we copy the following text:

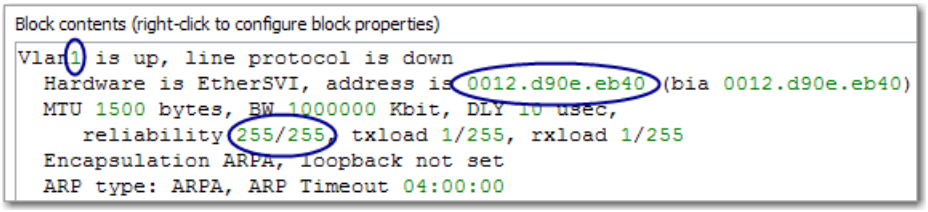
```
Vlan1 is up, line protocol is down
 Hardware is EtherSVI, address is 0012.d90e.eb40 (bia 0012.d90e.eb40)
 MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
   reliability 255/255, txload 1/255, rxload 1/255
 Encapsulation ARPA, loopback not set
 ARP type: ARPA, ARP Timeout 04:00:00
```

4 Paste the text into the **Block contents** box. iTest applies its built-in parsers (**Word**, **Number**, **MAC Address**, **Timestamp**, and so on) to color green the text that seems to be variable and to color black the text that seems to be static (words and punctuation). Each item of text in the response is now a ***token***.



5 Now we specify which tokens to ignore and which tokens to extract from any response, as follows:

- **Tokens that should have names:** Naming a token makes the value of the field available to be extracted from any response. (Analysis rules associated with test case steps return the value and then make the appropriate comparisons.) In our example, we'll name the circled tokens.

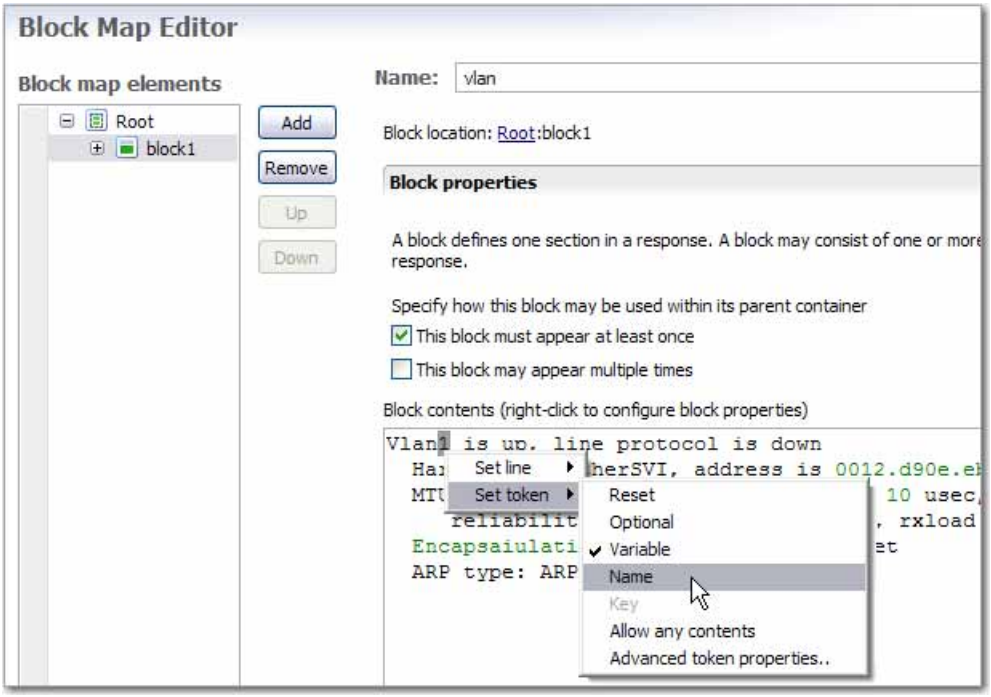


- **Tokens that will change from test to test, but that are also irrelevant to test case results:** For example, the **04:00:00** token in the last line was correctly identified by the Time parser as a value that might change from test to test (depending on device configuration). However, the value of the ARP Timeout token is not important for the kinds of tests that you will run. You will specify that such tokens can take on any contents.
- **Tokens that were *incorrectly* identified by iTest as not expected to change from test to test:** For example, the **ARP type** value in the last line is **ARPA** — iTest identified **ARPA** as a static word, but, in tests, this field can take on different values. You will need to specify that you would like to return and work with the value of this token in test cases.

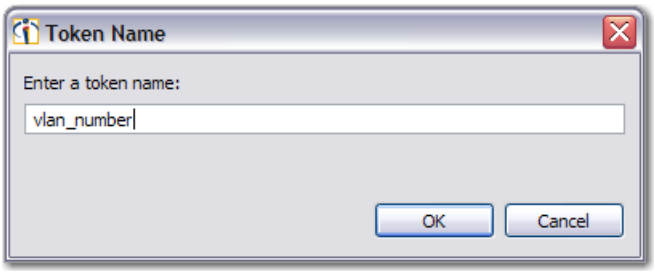
Configure the tokens in the block

The first token that we'll configure is the number that identifies the VLan whose data is being displayed. We'll name the token **vlan_number**.

1 Right-click the green 1 and select **Set token > Name**.

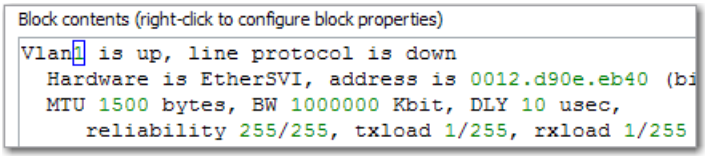


In the **Token Name** dialog box, type the name of the token and click **OK**.

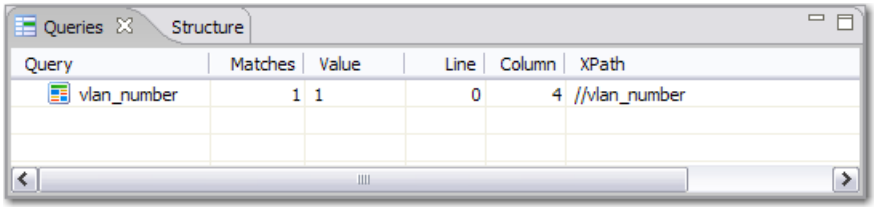


Notice the following changes:

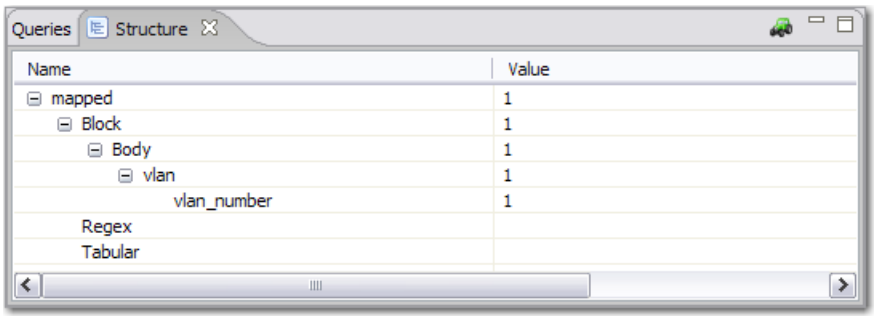
- The green **1** in both the Response view and the **Block** contents box is now enclosed in a blue box to indicate that it represents a token query that can return the value in this field from any response.



- The Queries view now displays one match with the `vlan_number` token query in line 0, column 4 of the sample response text. The value of the matching text is “1”.



- The Structure view now displays a match in the sample response text with the vlan_number token query and the value is “1”.



- The Issues view displays a mapping issue, but we can safely ignore issues while we’re putting the first configuration settings into place. More on this later.

2 Now let us configure the token that holds the MAC address of the device. Based on the format of the text (**0012.d90e.eb40**), iTest has identified it as a MAC address. Your task is to name the token so that, in any test case step that submits the `show interfaces` command, iTest can return the MAC address for further analysis or processing. This time, we’ll use a shortcut and double-click the green text to open the Token Name dialog box. Type the name of the token (let us use **device_mac_address**). Click **OK**.

Again, iTest updates the views to reflect the new token query.

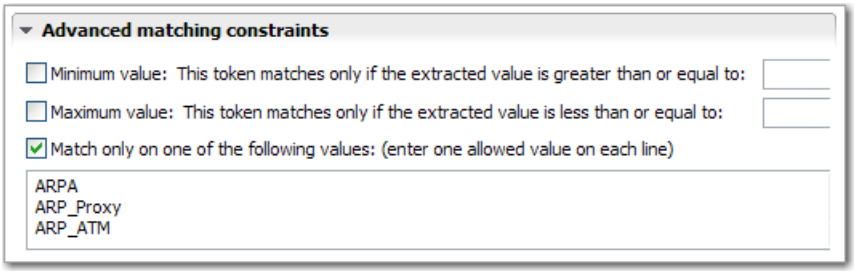
3 The value that you see for reliability is **255/255**. Notice that the two numbers are green to indicate that the parsers identified them as numbers (and therefore, likely to change from test to test). The slash character **/**, however, is black, indicating that it is a punctuation token and therefore unlikely to change its value from test to test. So it seems that our task is to configure two tokens. Let us call them the **reliability_score** and the **reliability_basis**.

Note Alternatively, you could select the full **255/255** text and name the token **mask**. Then, in the **Advanced Matching Constraints** section, check **Match only on one of the following values** and specify the value **255/255**.

---

4 iTest incorrectly identified **ARPA** in the last line as a static word. Now, let us configure it as the **arp_type** token because this field will take on different values that we want test case steps to return and analyze. Right-click the text and select **Set token > Advanced token properties**. The **Token properties** page opens. The name of the token appears in the **Name** box, and the token is correctly configured as **Variable** (the field can take other values that the **ARPA** text in the response sample). Now, in the **Advanced Matching Constraints** section,

check **Match only on one of the following values** and then type each value that you expect in responses, as in this example:



5 At this point, we have configured all of the named tokens in the response. The next step is to define the properties of the blocks in the response, as discussed in "Response Map editor: Block Map properties" on page 594.

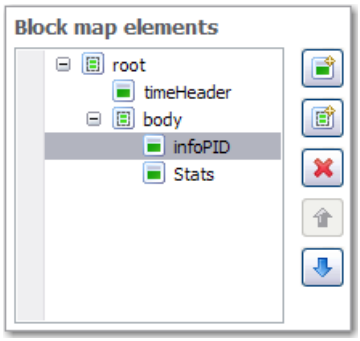
Response Map editor: Block Map properties

While defining a block map, you use the **Block Map** page to specify the properties for block map elements. This section describes all of the property settings.

Here's a quick discussion of the elements that make up the Block map (more detailed descriptions appear in "Overview: Properties of Block map elements" on page 589):

The **root** of the block is always named **root** and is the parent node for the containers and blocks that you structure the response text into.

- A **block** defines one section of a response — one or more lines of text — either related information or data that is grouped (often with a particular format).
- A **container** is a collection of blocks and child containers. Each container represents one section of a response. In the example, the **body** container holds two blocks: **infoPID** and **Stats**



Important By default, iTest tries to map responses by applying the containers and blocks in the order in which they appear in the tree. This can be an important consideration in your design. You can make use of **Move Up** and **Move down** to order the elements of the map as needed. See an additional note on ordering blocks within a container in "Defining a block" on page 596.

Defining the root of the block

The root has the following property settings:

General properties

Specify how to deal with minor mapping discrepancies	**Strict** (default): Declare an error and terminate mapping upon encountering any mapping error, including individual token mismatch **Loose**: Declare a warning upon encountering a token mismatch, but continue mapping the remainder of the response
Specify how the top-level blocks and containers should be used during mapping	**Ordered** (default): The flow of the response must match the blocks and containers in the listed order **Random**: The top-level blocks and containers may appear in the response in any order
Specify how to treat blank lines in the response and in block definitions when mapping	Use this setting to specify whether the mapping process should pay attention to missing or extra blank lines in responses. Lines that include tab and space characters (that is, whitespace characters) in an otherwise empty line are considered blank lines. If **Strict**, then blank lines that appeared in the response that was used to create the response map must also appear in actual responses and no extra lines can appear. If **Non-strict** (default), then no mapping errors result when blank lines or extra lines appear in responses regardless of whether or not they appear in the response that was used to create the response map.
Maximum search iterations	The maximum number of times that the iTest search algorithm should be applied to the response. The number of iterations needed to find a match is not related to the number of levels of containers. Rather, optional tokens, wild cards, and other options increase the complexity of the search. If this limit is exceeded without finding a match, then the step fails. Increase this setting only if the response map is complex or responses do not map well. **Caution**: We recommend that you do not modify this setting. Setting this property much higher than the default setting will likely result in stack overflow exceptions.
Maximum search time	Specify the longest time that iTest should spend attempting to find a match. If this limit is exceeded without finding a match, then the step fails.

Modify Contents

Click **Add block** to add a block to the root. The **Block properties** page opens.

Click **Add container** to add a container to the root. The **Container properties** page opens.

Defining a container

A container is a collection of blocks and child containers. In total, the container represents one section of a response.

Container properties

Depending on how it is configured, this section may repeat and/or appear multiple times at different locations within the response.

Specify how this container may be used within its parent container	This container must appear at least once. This container may appear multiple times
Specify how the blocks and containers within this container should be used during mapping	**Ordered**: The flow of the response must match the blocks and containers in the listed order. **Random**: The blocks and containers within this container may appear in the response in any order

Click **Add block** to add a block to the container. The **Block properties** page opens.

Click **Add container** to add a container to the container. The **Container properties** page opens.

Defining a block

A block defines one section of a response — one or more lines of text — either related information or data that is grouped (often with a particular format).

Name

Give the block a meaningful **Name**.

Important Do not give a block a name that you plan to give to a token within the block. That is, do not give related response maps, containers, blocks, and tokens the same name.

Block contents

Paste the text that makes up the block into the **Block contents** text box. iTest parses the response to identify likely tokens.

Block properties

The **Block** properties specify how the block may appear within its parent container — it might or might not appear at all. If it does appear, it might or might not appear more than once.

This block must appear at least once	Checking the box (default) indicates that the block of text is required to be in the response. See the **This block may appear multiple times** property. See “Optional blocks: How block order affects the mapping process” following this table.
This block may appear multiple times	Uncheck the box (default) to indicate that the block can appear more than one time within the response. Check the box to indicate that the block can appear at most one time within the response.

Optional blocks: How block order affects the mapping process

In the example we’ll discuss here, a command can return either of the following responses:

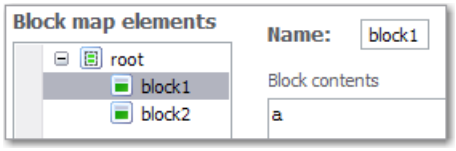
a

or

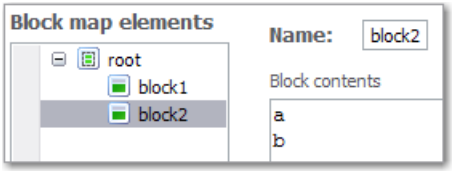
a

b

To allow for these variations, you might expect that you can define two optional blocks in the map, (that is, for both **block1** and **block2**, the **This block must appear at least once** property is unchecked).



and



Because iTest tries to map responses by applying blocks in the order in which they appear in the tree, however, a response of the form

a

b

would *always* be mapped by **block1** (`"a"` matches `"a"`). For this reason, you should order block definitions so that the most inclusive block is applied to the response first. In this example, we would place **block2** *before* **block1** in the **Block Map elements** tree to ensure that both forms of the response will be mapped.

Line properties

Each block is a collection of ordered lines of text. Each line is a collection of tokens.

This line is optional within the block. If you check the box, then, if the line matches a line in the response, then tokens within the line will be mapped. If the line is absent, no error will be declared.

Alternatively to opening the **Line Properties** section of the page, you can right-click a line and select **Set Line** to configure the properties of the line, as follows:

Reset	Remove the current token definitions so that you can define tokens as needed.
Optional	This selection checks the **This line is optional within the block** property for the line.
Allow any contents	Specify that the line can contain any text or no text.
Advanced line properties	Open the **Line Properties** section of the page.

Token properties

Values for tokens that are named will be returned and available for postprocessing. The **Token** properties specify how the token will appear in the response.

Token name	Naming a token makes the field's value available to be returned from any response. You can associate an analysis rules with a test case step to return the value of the named token and then make the appropriate comparisons (the value is not zero or is the same as another value, and so on). Naming a token is valuable in the case that the token's value represents a Pass/Fail condition. For example, a step's analysis rule can test whether the token named **PortStatus** (with allowed values **up** and **down**) can pass the test case if the value of **PortStatus** is **up** and fail the test case otherwise. Once you assign a name to a token, the token value is enclosed in a blue box in the Response view. In addition, once you generate a query for it (either manually or by having iTest auto-populate it) the token is listed by name (with its associated query) in the Queries view.
Wildcard	The token can represent any string (typically a fragment of a line) that you would like to parse as a single entity. This setting is useful when you want to allow for values that are not being parsed correctly by iTest's built-in parsers. **Wildcard** tokens are restricted to one line, but can appear anywhere in the line. If a **Wildcard** token is encountered while mapping, it can map to zero or more tokens of any type in the same line from the actual response. You can use the **Optional** property on a Wildcard to indicate whether it needs to match to zero or more versus one or more. This token will match any number of contiguous tokens with any values on the line. iTest displays values for the following two items: **Value**: A string that contains a representative value that is displayed when showing the mapped response **Parser**: The name of the parser that identifies the token in the response string. For example, if the string "Thu 18-Sep-03 18:42" appears in a response, the **Http Timestamp** parser will identify the string as a timestamp. The parser associates a token with the identified timestamp value and uses the appropriate value representation to store its value.
Optional	This token may or may not be present in a line that matches Check **Optional** if it is acceptable for no value to appear for the token. This setting is useful for values that might or not appear in a response.
Variable	Check **Variable** if it is acceptable for a variety of values to appear for the token. The matching token can take other values that can be parsed with the same parser. This setting allows values like timers and statistics to vary without causing mapping failures.
Key	This token appears in a repeating block and should be used as a key when auto-generating token aliases When True, use the value of the token as the key to identify the instance of a repeating block in the response that contains the value that you want to return. For an example, see **Configuring a Token as a Key (Index) to Other Tokens in the Response** iTest generates a query for each combination of a key token and all other tokens in the response map.

Advanced matching constraints

Minimum value	This token matches only if the returned value is greater than or equal to: For numeric tokens, the minimum allowed value for the token. See **MaxValue**. For tokens that can take on any of a set of specified values, see **AllowedValues**. **Note**: Use an analysis rule to make Pass / Fail decisions.
Maximum value	This token matches only if the returned value is less than or equal to: For numeric tokens, the maximum allowed value for the token. See **MinValue**. Use 0 to represent "any value allowed". For tokens that can take on any of a set of specified values (text or numeric), see **Match only on one of the following values**. **Note**: Use an analysis rule to make Pass / Fail decisions.
Match only on one of the following values	(Enter one allowed value on each line) The collection of values that this token can take on. For example, the **PortStatus** token might take on either of the two values **Up** or **Down**. The most common use for the **AllowedValues** property is for tokens that can take on any of a set of specified values (text or numeric). In the text box, type the allowed values, one per line. When you have finished entering values, press Ctrl-Enter. For tokens that can take on numeric values within a range, see **MinValue** and **MaxValue**. **Note** Use an analysis rule to make Pass / Fail decisions.

When there are multiple response formats for a particular command

Some commands can return different responses under different conditions. For example:

- A command returns five columns of data on one platform and only three columns on a different platform
- Platform-independent teams must test features across several platforms and responses can vary widely

To map the responses, create multiple response maps — one for each form of the response.

Key strategies

- Create a unique response map for each form of the response and let iTest choose which response map to apply.
  - Keep the maps together in a response map library.
  - Configure identical **Applicability** settings for each response map.
- Ensure that iTest cycles through all applicable maps when attempting to map a response. With this setting, if mapping fails for one map, iTest generates an error and then tries the next applicable map. If any map ultimately succeeds in mapping the response, then iTest does not publish the error message.

- For table maps: On the **Table** page of the Response Map editor, we check the **Required: Generate an error if the table is not found at least once** check box.
- For pattern maps: On the **Pattern** page of the Response Map editor, we check the **Generate an error if no matches are found** check box.
- For block maps: On the **Block** page of the Response Map editor, for all blocks and containers, we check the **This block/container must appear at least once** check box.

Mapping JSON responses

iTest understands JSON structured data, so it can typically map a JSON response without requiring a response map. The structured data root for JSON responses is: **mapped/Json/**

A simple JSON object `{name:"value1"}` is structured: **mapped/Json/name**

A simple JSON array `["value1", "value2"]` is structured: **mapped/Json/item[]**

The iTest JSON mapper is more tolerant than the JSON specification.

Note Empty ("") JSON object name will be replaced with the special word, "`iTestEmptyXmlKeyName`" in structured response.

- An extra , (comma) may appear just before the closing bracket.
- The null value will be inserted when there is, (comma) elision.
- Strings may be quoted with ' (single quote).
- Strings do not need to be quoted at all if they do not begin with a quote or single quote, and if they do not contain leading or trailing spaces, and if they do not contain any of these characters: { } [ ] / \ : , = ; # and if they do not look like numbers and if they are not the reserved words **true**, **false**, or **null**.
- Values can be separated by ; (semicolon) as well as by a , (comma).
- Numbers may have the 0- (octal) or 0x- (hex) prefix.
- Comments written in the slash-slash, slash-star, and hash conventions are ignored.

Mapping TL1 responses

See “Mapping TL1 responses” on page 1216.

Setting preferences for response mapping

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Response Mapping.**

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Spirent > Response Mapping

Maximum number of queries to evaluate	Default: 500 Specify how many queries that a rule should evaluate before displaying a warning that no match was found and that the maximum number of queries had been tried.
Maximum number of queries in group to evaluate	Default: 100 Specify how many queries in a regular expression group that a rule should evaluate before displaying a warning that no match was found and that the maximum number of queries had been tried.
Enable automatic response map generation when response is loaded into Response view	Default: selected By default when a response is loaded to Response view a response map is generated. Automatic response map generation may take some time for long responses. To avoid long delays on viewing responses, clear the check box to disable automatic response generation, if not required.

Spirent > Response Mapping > Block Mapping

Automatically expand/collapse block property page sections	Default: checked Because there are several sections on the **Block Property** page, you can use this setting to cause sections to collapse when not in use.

Spirent > Response Mapping > Table Mapping

Banner color	Click a color to open a dialog box that enables you to specify a different color.
Banner match color	
Body color	
Odd column color	
Even column color	
Column end marker color	
Footer color	

CHAPTER 25

Filtering Unwanted Text from Responses

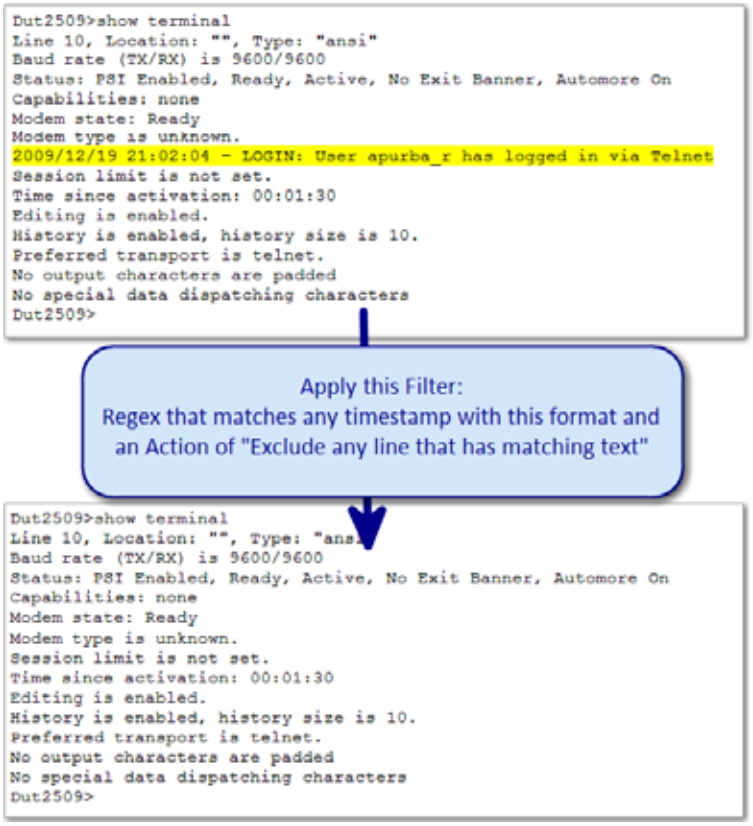
Overview: Response filtering

You can use ***response filters*** to remove unwanted text from a response after a step has executed and before iTest applies analysis rules. The “filtered” response (the portion of the response that remains) is typically cleaner to read and easier to map and to understand. As a result, the sample responses that you use to create Response maps are simpler, and test reports are more readable.

Filtering a response is also useful when an otherwise normal response is corrupted by extraneous text. In the following example, an autonomous log message from the terminal server is intermixed within the stream of the response from the system console (we highlighted the log message in the “unfiltered” response at the top). Because an unpredictable text string appears in a random location, a response that is normally easy to map cannot be mapped.

A response filter can solve the problem. Because the log messages have a well-defined structure, you can design a regular expression that will match any message (perhaps matching the timestamp format). The regex and an associated “exclude lines with matching text” action

can therefore act as a “filter” to remove all log message text from any response. The result is the “filtered” response at the bottom. Now the response map can do its job.



Note The text that was discarded is not actually lost; it is simply not displayed in the Response view or in reports. You can click **Filtered/Unfiltered response** in the Response view to toggle the view between the filtered and the unfiltered response. In addition, you have the option to save any excluded text into the structured data for the step.

When should you filter a response?

You will find filtering to be useful in the following situations:

- The response contains a lot of irrelevant text, and it would be easier to analyze and display only a portion of the response and to ignore the rest.
- The device produces logging messages that are mixed into the output (common when using a terminal server and with TL1 devices). The messages appear as separate lines. You might want to analyze the messages in a different step, but you need to filter out the messages so that you can define a response map for the base response.
- The device produces XML output, but the output includes non-XML headers and footers that corrupt the XML (or HTML) mapping. Filtering can remove the headers and footers before you apply the queries.

How response filtering works

- **Define the filters**

  First, you specify the **Pattern** of text that you are looking for in the response (the pattern to match). You can specify text with a wildcard character, a regular expression, or a case-insensitive text match.

  Next, you specify the **Action** — what to do when a match occurs — for example, discard any line that contains a match, or discard every line that does not contain a match, or include everything up to the matching text and discard everything else, and so on.

  - You can define multiple filters for a particular response
  - You can specify that text that is discarded by a filter should be added to the structured data for the step
  - You can define filters for a session profile, for an executable step in a test case, and for the response map used by a step

- **Then, during execution...**

  1. iTest applies the filters to the response. “Applying a filter” means:
     a. One line at a time, check for matches with the **Pattern**
     b. If a match occurs, perform the specified **Action**
  2. iTest applies the filters in the following order to the text that remains in the response:
     a. filters defined in the session profile, in the order that they are listed
     b. filters defined for the step, in the order that they are listed
     c. filters defined in the response map, in the order that they are listed
  3. The resulting filtered response for the step contains only the lines that were not excluded. This is the response text that analysis rules check and that appears in report.

Defining response filters

1. Open the appropriate editing tool:
   - To define a filter for a step: In the Test Case editor, select the step. In the **Step Properties** section, open the **Response Filters** page.
   - To define a filter for a session profile: In the Session Profile editor, click the **Response Filters** tab.
   - To define a filter for a response map: In the Response Map editor, click the **Response Filters** tab.
2. Optional. Modify whether existing filters are inherited (does not apply to response maps — response maps cannot inherit filters). By default, the **Include inherited values** property is checked, therefore:
   - By default, a step inherits the filters that are defined in the session profile associated with the step’s session.

- By default, a session profile inherits the filters defined in the session profiles that the profile is based on.

To modify or delete inheritable filters, uncheck the **Include inherited values** check box. The list now displays all inherited filters. To edit a filter definition, select it in the list and then change settings as described in Step 4.

Note Because the formerly inherited filters are now defined in the current step or session profile, the filters are not updated when the filters in the reference session profile change.

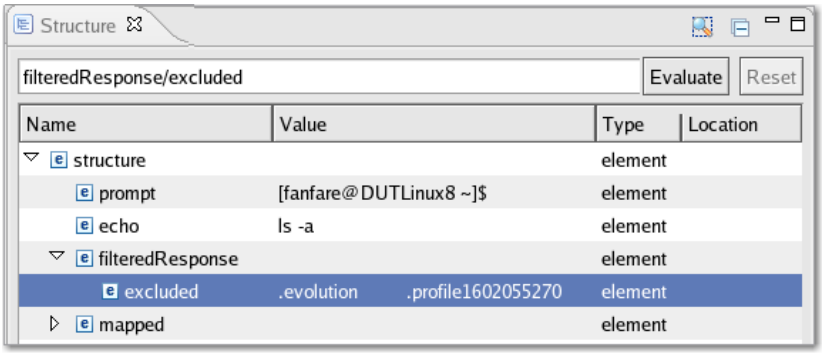
3 For steps and for session profiles only: To enable you to add filter definitions, check **Include additional values from list**.

4 Click **Add** to add a filter and then specify the following properties for the new filter.

Tip To filter multi-line text:
1. Set the **Action** property to **Include matches of the pattern**
2. Ensure that the **Pattern** property includes a RegEx that will hit the multi-line text

Name	Specify a meaningful name for the filter. For example, **deleteLogMsgs** or **includeOnlyPortStatus**
Action	Select one of the following methods for applying the pattern while filtering a response. • **Include** means that the matching text remains as part of the response. • **Exclude** means that the matching text is discarded: If you choose to exclude data, you have the option to add the excluded text to the structured data for the step. See the **Add discarded text to structured data** property. Include only lines matching the pattern Include only lines containing matches of the pattern Exclude lines matching the pattern Exclude lines containing matches of the pattern Include matches of the pattern (each on a separate line in the output) Exclude matches of the pattern found within lines Include lines starting with the first line matching the pattern until the end Include lines up to but not including the first line matching the pattern Include lines up to but not including the first line containing the pattern Include lines up to and including the first line matching the pattern Include lines up to and including the first line containing the pattern
Pattern type	Specify how to interpret the pattern that you specified for the **Pattern** property: Case Insensitive, Wildcard, or Regex Default: Wildcard
Pattern	Specify a string that represents the text that you are looking for within the response. You specify whether to include or exclude matches (or to perform other actions) using the **Action** property. You can use field replacements in the pattern text

Add excluded text to structured data	Optional Check the box to add the excluded data to the structured data. In the structured data, the text is added to the **filteredResponse** element (**filteredResponse** is parallel to the **prompt** element). The data is added in the element that you specify for the **Excluded data tag** property. The **Value** element holds the excluded text with one "item" for each excluded line. An example appears below this table.
Excluded data tag	Required if you check **Add excluded text to structured data**. Specify the XML tag that should identify the data that you are adding to the structured data. The data is inserted into the structured data at this XPATH location relative to the **filteredResponse** tag. In the example that appears below this table, we named the tag "**excluded**". You can use field replacements in the text that defines the tag. Default: [empty]



- Add as many filters as needed.
- Filters are applied in the listed order. To change the order, select a filter in the list and use **Move up** and **Move down** .
- To delete a filter definition, select a filter in the list and click **Remove** .

Comparing the filtered and unfiltered version of a response

The Response view includes the **Filtered/Unfiltered response** toolbar button that allows you to toggle between viewing the filtered response and the unfiltered response. By default, the view displays the filtered response.

Note You cannot add an analysis rule while viewing an unfiltered response.

CHAPTER 27

Events: Taking Action when a Particular Event Occurs During Execution

Events

You can add considerable power to a test case by specifying that, whenever a particular ***event*** occurs during execution, iTest should perform a certain ***action***. For example, you can configure that when a step times out (an **OnStepTimeout** event occurs), iTest should perform the following two actions:

- Perform a **DeclareExecutionIssue** action: Issue an execution issue with a severity level of **Error** with associated execution message text "**Step has timed out.**"
- Perform a **FailTest** action: Set the test result to **Fail** and continue to execute.

Where you can configure Events

iTest events perform particular actions by default. Based on where they are defined, events are applied as follows:

- The **Global Events** page on the Session Profile editor (also called the **New Session** page): Actions that you define for events are applied to any step in a session that is based on the session profile.
- The **Events** properties section for the selected devices in the Testbed editor. Actions that you define for events are applied for any step that uses the selected devices.
- The **Events** properties section for a selected step in the Test Case editor (described in "Step Properties section: Events properties group" on page 181). Actions that you define for events are applied to the selected steps.
- The **Global Events** page on the Test Case editor. Actions that you define for events are applied to any step in the current test case.

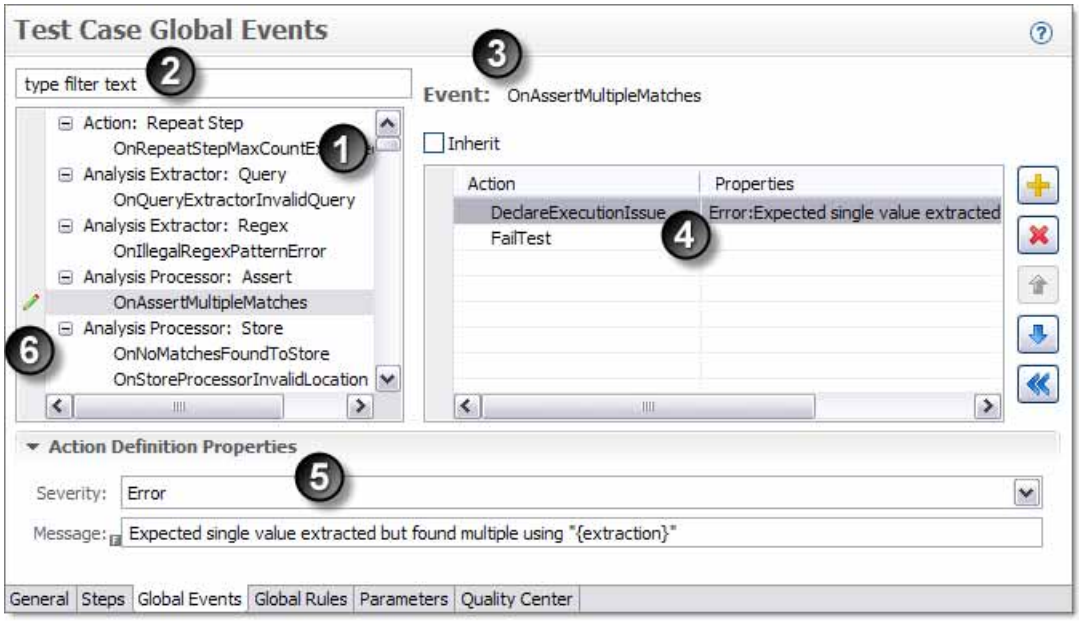
The process of configuring the actions to take for an event (regardless of where you set the properties) are described in "Configuring Events: The Global Events page" on page 603.

Configuring Events: The Global Events page

The **Global Events** page appears on the Test Case editor and on the Session Profile editor.

- On the **Global Events** page in the Test Case editor, you specify the actions that should take place whenever particular events occur during execution of any step in the test case.

- On the **Global Events** page in the Session Profile editor, you specify the actions that should take place whenever particular events occur in the session that started with the session profile.



1 List of events

Events are listed in functional groups. Here, in the **Analysis Processor: Assert** group of events, we selected the **OnAssertMultipleMatches** event. The full list of available events appears in “Event definitions” on page 606.

2 Filter

To help you to locate a particular event more quickly, you can limit the events that appear in the list by typing part or all of an event name into the text box. Only event names that match the filter are displayed in the list. The * and ? wildcard characters are supported.

To reset so that no filter is applied, delete all text.

3 Event section

When you select an event, the **Event** section lists the actions that will execute if the event occurs. Some events have default actions.

4 Actions for the selected Event

The default **Actions** for the selected **Event** appear in this list. (You can review the full list of events in “Actions on events: Definitions” on page 626.)

Actions are taken in the order in which they appear in the list (the first action in the list occurs first). Use **Move Up / Move Down** to reorder the list.

5 To edit an Action:

a Un-check **Inherit** to enable non-inherited (non-default) settings.

b Click **More** » to display the properties of the selected action in the **Action Definition Properties** section.

c Edit the **Action Definition Properties** settings as needed.

6 Non-default settings

If you change an Event to a non-default setting, the pencil icon appears next to the step in the **Steps** grid, the **Events** property group, and the **Event name**. This will help you at a later time to locate custom settings.

Note The **Pencil icon** ( ) indicates that a value has been explicitly provided by you. It is a reminder for you to verify the new value in the **Step properties** (in the Step properties group hierarchy). That is, in a session or step properties, the **pencil icon** indicates that there may be additional/different values set for that step/session. For example, an IP Address is a required argument. If the value is empty, an error indication displays on the action line. When you enter a value (it is not empty), a **pencil icon** displays indicating that the value is not equal to the default value and requires verification.

Precedence of Event settings

- Actions that are defined on the Session Profile editor's **Global Events** page are considered as being properties for any step in a session based on the profile.
- When session profile A is based on session profile B, then session profile A's settings take precedence.
- Actions that are defined on the **Global Events** page for a test case take precedence over actions defined for a session profile.
- Actions that are defined in the step properties for a particular step take precedence over actions defined on the **Global Events** page for a test case.

Global Events list

The **Global Events** list includes all possible events for the current step. When you select an event, the actions that should be taken when the event occurs appear in the **Action** list. All events have default actions.

Editing actions

- Actions are performed in the order in which they appear in the **Actions** list.
- Many events have default actions that are listed in the **Actions** list
- To modify the actions for an event, you must uncheck **Inherit**. To revert any Event to its default actions, check **Inherit**.

Add	Add a new action after the selected actions.
Remove	Delete the selected actions.

Move Up **Move Down**	Actions are taken in the order in which they appear in the list (the first action in the list occurs first). Use these buttons to reorder the list.
More	View the properties that you can set for the selected action. You must un-check **Inherit** to enable the button.

Configuring events for steps

To define the set of actions to perform when a particular event occurs while a step is executing:

1. On the **Steps** page, select the step.
2. On the **Step Properties** page, select **Events**.
3. Now follow the instructions for configuring a Global event: “Configuring Events: The Global Events page” on page 603.

Event definitions

Actions that can be performed when an event occurs are described in “Actions on events: Definitions” on page 626.

Group / Event	Description
Action: Repeat Step	
OnRepeatStepMaxCountExceeded	Occurs when the execution time of "RepeatStep" action has exceeded the specified **MaxCount** value. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Maximum repeat count of {0} has reached the limit
Analysis Extractor: Query	
OnQueryExtractorInvalidQuery	Occurs when performing the response mapping during executing the test case, an invalid query was found (either because the query string itself is not valid or the query does not match the runtime response map definition). **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Invalid query: {0}
Analysis Extractor: Regex	
OnIllegalRegexPatternError	Occurs when the regular expression is empty or has a syntax error. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Invalid regex pattern: {error}
Analysis Processor: Assert	

OnAssertMultipleMatches	Occurs when multiple values extracted for the assertion in analysis rule. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Expected single value extracted but found multiple using “{extraction}”

Analysis Processor: Store

OnNoMatchesFoundToStore	**Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: No matches found to store in variable "{0}"
OnStoreProcessorInvalidLocation	**Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Unable to store result at specified location: {0}

Application: Agilent N2X

OnAgilentCommandFailed	—
OnAgilentConfigurationLoadFailed	—
OnAgilentSessionDisconnected	—

Application: HTTP

OnAuthenticationFailure	Occurs when basic authentication information is absent. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Authentication error for URL {0}
OnCommandFailed	Occurs when an I/O exception (not an authentication error) occurs. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Connect to {0} failed, reason: {1}
OnConnectionRefused	Occurs when connection exception is thrown **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Host refused connection on URL {0}
OnFileNotFound	Occurs when a URL is not found **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: URL {0} is not found
OnServerUnreachable	Occurs when the host designated by URL is not reachable **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Host for URL {0} is unreachable

OnUnavailableEncoding	Occurs when the charset is not set in the charset property page. **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Encoding {exp} unavailable, using {act} instead

Application: IxLoad

OnIxLoadCommandFailed	Occurs when executing a test case step failed **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Command {0} failed: {1}
OnIxLoadConfigurationLoadFailed	Not implemented
OnIxLoadSessionDisconnected	Not implemented

Application: IxNetwork

OnIxNetworkCommandFailed	Occurs when executing a test case step failed. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Command {0} failed: {1}
OnIxNetworkConfigurationLoadFailed	Occurs when executing "load configuration" command failed or the session loads configuration file failed. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Failed to load configuration: {0}
OnIxNetworkSessionDisconnected	Occurs when executing a test case step and the response contains "not connected to IxNetwork". **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {message}

Application: Ixia Traffic

OnIxiaCommandFailed	Occurs when executing a test case step failed **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Command {0} failed: {1}
OnIxiaConfigurationLoadFailed	Occurs when the configuration file fails to load. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Failed to load configuration: {0}
OnIxiaSessionDisconnected	Not implemented

Application: Mail

OnInvalidMailProperties	Occurs when the mail properties are invalid, for example: an empty SMTP URL. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid Mail properties: {0}
OnMailSendFailure	Occurs when the mail 'send' action has issues **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Failed to send mail: {error}

Application: Process

OnProcessLaunchFailure	Occurs when the process cannot be started **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error while starting process: {command}
OnProcessLimitExceeded	Occurs when the started process number exceeds the limitation. Default is 100 **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Process limit ({limit} instances) exceeded

Application: SNMP

OnSnmpErrorLoadingMibFiles	Occurs when any of the following conditions is encountered: • Undefined MIB folder • Specified MIB folder URI is invalid • Could not open specified MIB folder • No files found in specified MIB folder • Could not load or parse MIB file • The **Strict MIB parsing** property is enabled in the SNMP session profile, and an error while resolving a MIB object in the MIB file fired the event. (You can disable the **Strict MIB parsing** property in the SNMP session profile and then restart the test case.) **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Error while loading MIB files: {0}
OnSnmpInvalidAction	Occurs when executing **getTable** action on a non-table node on the MIB tree **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: The requested action "{action}" is not appropriate for this MIB variable "{nodeName}":{reason}

OnSnmpInvalidCommandSyntax	Occurs when the argument of **set** command is blank **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Invalid command syntax: {reason}
OnSnmpOpenFail	Occurs when any of the following conditions is encountered: • Specified trap port is not a number or is not between 1 and 65535 • A daemon is already running on the specified port with different settings • The third-party iReasoning SNMP API throws an error for new SnmpSession(target) **Default actions:** • **FailTest** • **AbortExecution** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error while opening SNMP session: {0}
OnSnmpRequestFail	Occurs when any of the following conditions is encountered: • Loop or duplicate MIBs found in **get**/**getNext**/**getTable**/**walk** action • SNMP session is closed before **get**/**getNext**/**getTable**/**walk**/**set** action is finished • New MIB value is invalid for **set** action • Cannot find the MIB definition in the loaded MIB files for **set** action **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: SNMP request failed: {0}
OnSnmpStartTrapListeningFail	Occurs if the test executes a **WaitForTrap** action before a trap daemon is successfully started. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error trying to start listening for SNMP traps: {error}
OnTrapPortBindFailed	Occurs in the following cases: • Specified trap port is not a number or is not between 1 and 65535 • A daemon is already running on the specified port with different settings **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: SNMP trap daemon failed to bind to port {port} : {error}

Application: SmartBits

OnSmartBitsCommandFailed	—
OnSmartBitsSessionDisconnected	—

Application: Spirent TestCenter

OnSpirentTestCenterCommandFailed	—

OnSpirentTestCenterConfiguration LoadFailed	—
OnSpirentTestCenterSessionDisc onnected	—

Application: Swing

OnSwingGeneralError	Occurs when general errors happen in a Swing session **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid swing action: {0}
OnSwingHeadless	**Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {error}
OnSwingExecutionError	Occurs when swing session execution errors happen **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Action failed: {0}

Application: Syslog

OnSyslogCommandFailed	Not implemented
OnSyslogSessionDisconnected	Not implemented

Application: Tcl Shell

OnTclshCreateInterpreterFail	Not implemented

Application: Web

OnClearCacheError	Occurs when new issues are introduced after clearing cache. **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnClearCookiesError	Occurs when new issues are introduced after clearing cookies. **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnInvalidMouseEventArgument	Occurs when mouse event arguments in step property are not valid. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {error}

OnWebExternalBrowserError	For Web session with external browser: Occurs when cannot connect to the browser **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error communicating with external browser: {0}
OnWebHeadless	Not implemented
OnWebHttpError	For Web session with external browser: Occurs when HTTP status code is higher than 400.
OnWebInvalidProperties	Occurs in the following cases: • Web session tries to get form map library but fails • The **open** step is executing but the application property is not of the **open** step. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid web properties: {0}
OnWebOpenExternalBrowserFail	For Web session with external browser: Occurs when system tries to connect to the external browser but fails. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error attempting to open connection to external browser: {0}
OnWebOpenFail	Occurs when web session fails to open web editor for the internal session **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {error}
OnWebStepAborted	Occurs in the following cases: • Session step execution is canceled; • Session step is trying to get the target from web page but the **XPCOMException** is thrown **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Web step aborted
OnWebStepSyntaxError	This event will occur for any issue generated during Web session execution Occurs in the following cases: • For a selectWindow step, no window is specified • For a popup window, no popup wait time is specified • For a step that needs a target, no target is specified • For a step that needs a target, no target wait time is specified • When loading a new web page, no popup wait time is specified **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid web step: {0}

OnWebTargetNotFound	Occurs when a specified target is not found on the web page (either because it is not on the page or it has not appeared during the period specified for the **Maximum time to wait for a target** property). No default actions. **name** variable value: <targetName> **message** variable value: "Specified target {name} was not found on the browser page {name} at {URL}"

Application: Wireshark

OnWiresharkCommandFailed	Occurs when a command fails **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Command {0} failed: {1}

Data extractor: XML

OnInvalidXPathQuery	Not implemented
OnResponseIsNotValidXml	Not implemented

Data extractors

OnNoMatchesFound	Occurs when the search string (as defined by query or regular expression) is not found **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: No matches found for {type}: {info}
OnUnknownExtractorError	Occurs while handling an analysis rule when no corresponding extractor is defined **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Extractor {extractor} does not exist or is obsolete
OnEmulationInvalidStructuredDat a	Occurs when the structured data for an emulated response (after substitution of field replacement text) is invalid. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Invalid structured data: <errorText>
onEmulationSampleNameNotFou nd	Occurs when a step that is configured to use a sample response as the emulated response cannot locate a sample in a response map with the specified **Sample name**. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: No sample response with Sample name <specifiedName> was found

OnEmulationSourceNotFound	Occurs when the **External source** property is not blank and the source is not found or fails to open. **Note** When the source response map or response map library is not found, then normally a validation error with a problem marker results. **Default action** **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Emulation source not found: <errorText>

Execution

OnAbortExecution	Default action: **AbortTest**
OnAbortTestAction	Occurs each time an **AbortTest** action is performed — even if the test result is currently **Abort** and the **AbortTest** action does not change the result. Often used, for example, to perform a **CallProcedure** or **PauseExecution** action. **Note** If the **OnAbortTestAction** event occurs because an **AbortTest** action resulted in an earlier **OnAbortExecution** event, then the current **OnAbortTestAction** event cannot perform a **procedureCall** action (because execution has been aborted). No default actions.
OnAbortTestResult	**Default action** **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **“**Error: {message}”
OnDeserializationWarning	Occurs when the user attempts to open a iTest file with deserialization issues, for example, a document in a format that is no longer supported by the current version of iTest. **Default action** **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of “Warning during deserialization: {warning}”
OnExecutionCompleted	Occurs immediately after the test case finishes executing. Does not execute deferred actions (see “Deferred actions” on page 625). **OnExecutionCompleted** can initiate only immediate actions (actions that are not deferred, as described in “Deferred actions” on page 625). **Default action** **DeclareExecutionIssue** with a **Severity** of **Information** and with a default **Message** of: “Information: {message}”
OnExecutionStarted	Occurs immediately before the first step in the test case is executed. Often used, for example, to call an initialization procedure. **Default action** **DeclareExecutionIssue** with a **Severity** of **Information** and with a default **Message** of: “Information: {message}”
OnExecutionTimeout	Occurs when the time period specified for the **Execution Time Limit** property is exceeded (see “Execution Behavior” on page 166). Does not execute deferred actions (see “Deferred actions” on page 625). **Default actions** • **FailTest** • **AbortExecution** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Test execution has timed out

OnFailTestAction	Occurs each time a **FailTest**, action is performed — even if the test result is currently **Fail** and the **FailTest** action does not change the result. Often used, for example, to perform a **CallProcedure** or **PauseExecution** action. No default actions.
OnFailedTestResult	**Default action** **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: {message}"
OnFatalThreadError	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Fatal Error: {0}" • **FailTest** • **AbortThread**
OnInternalError	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Internal Error: {0}" • **FailTest**
OnInterpreterError	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: {error}" • **FailTest**
OnInvalidArguments	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Error processing description: {0}" • **FailTest** • **AbortExecution**
OnInvalidEncryptedCommand	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: The command must evaluate to a single encrypted string (and nothing else additional)" • **FailTest**
OnInvalidStepPropertyType	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Application properties associated with the step are of a type that is invalid for this step: {0}" • **FailTest** • **AbortStep**
OnInvalidTestCase	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Fail to load test case: {message}" • **FailTest** • **AbortStep**
OnInvalidTestbed	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Testbed is invalid: {0}" • **FailTest**

OnLoadingTestCaseFailure	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Error loading test case {testcase}: {errorMsg}" • **AbortExecution**
OnPassIfNotAlreadyFailedAction	Occurs each time a **PassTestIfNotAlreadyFailed**, action is performed — even if the test result is currently **Pass** or **Fail** and the **PassTestIfNotAlreadyFailed** action does not change the result. Often used, for example, to perform a **CallProcedure** or **PauseExecution** action. No default actions.
OnPassTestAction	Occurs each time a **PassTest**, action is performed — even if the test result is currently **Pass** and the **PassTest** action does not change the result. Often used, for example, to perform a **CallProcedure** or **PauseExecution** action. No default actions.
OnPassTestResult	**Default action** **DeclareExecutionIssue** with a **Severity** of **OK** and with a default **Message** of: "OK: {message}"
OnPreExecutionCompleted	Occurs immediately before test case finishes executing. Often used, for example, to call a "cleanup" procedure. No default actions.
OnProcedureEnter	Occurs when a procedure or QuickCall starts execution. No default actions. **Caution** Do not add a **CallProcedure** action — this results in an infinite loop. **name** variable value: <procedureName> **message** variable value: **"**Entering procedure {name}"
OnProcedureExit	Occurs when a procedure or QuickCall exits execution. No default actions. **Caution** Do not add a **CallProcedure** action — this results in an infinite loop. **name** variable value: <procedureName> **message** variable value: **"**Exiting procedure {name}"
OnReportingTestbed	**Default action** **DeclareExecutionIssue** with a **Severity** of **Information** and with a default **Message** of: "Information: Testbed used in execution: {testbed URL}"
OnStackOverflow	**Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **"**Error: Execution causes stack overflow" • **AbortExecution**

OnStepTimeout	Occurs when step execution time exceeds the **Timeout for this step** property setting (see “Step Properties section: Timing properties group” on page 178). • If the step is still executing, iTest aborts the step and does not process any analysis rules • If the step is executing an analysis rule, then iTest finishes the current rule and then skips all other analysis rules **Note** iTest does not enforce timeout settings for **call** steps (for procedures or QuickCalls). **Default actions** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: **“**Error: Step has timed out” • **FailTest**
OnTestResultChange	Occurs each time the test result changes. • From **Indeterminate** to **Pass**, **Fail** or **Abort** • From **Pass** to **Fail** or from **Fail** to **Pass** • From **Pass** or **Fail** to **Abort** Often used, for example, to perform a **CallProcedure** action. The procedure must determine the current test result (**Pass**, **Fail**, **Abort**, or **Indeterminate**) using either a **summarize** step or the **[info status]** command. No default actions.
OnThreadEnter	Occurs when a thread starts execution. Deferred steps are called after the first step in the thread has executed. If the thread step is a **call** step, deferred steps are part of “call” step. Otherwise, they are executed after the first step in the thread. If you use **OnThreadEnter**, then you should create a **comment** step for the thread (if the asynch step is not a **call** step) and nest the actual steps under the **comment** step. **name** variable value: <threadID> **message** variable value: **“**Entering thread {name}” No default actions.
OnThreadExit	Occurs when a thread exits execution. **name** variable value: <threadID> **message** variable value: **“**Exiting thread {name}” No default actions.
OnWarningTestResult	Occurs when test case pass/fail criteria are nt set for the test case **Default action** **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of “Warning: {message}”

Execution: Non-Session Actions

OnExecAssertionError	Not implemented

OnExecCallInvalidUri	Occurs when calling a procedure with invalid URI (typically the URI was manually typed by the user). **Default actions:** • **Abort step** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Call to an invalid test case uri: {0}
OnExecCallUnknownProcedureError	Occurs when calling a procedure that cannot be found by the specified URI **Default actions:** • **Abort step** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Call to unknown procedure: {0}
OnExecGoToLabelNotFound	Occurs when the goto label does not refer to a valid step label **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnExecIllegalActionError	Occurs when calling an action that is not valid for current state. For example, calling the **break** or **continue** action outside of a **for** loop. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Illegal action {action} : {explanation}
OnExecInvalidCallArguments	Occurs when calling the procedure with invalid arguments **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error processing procedure call arguments: {0}
OnExecReadFileError	Occurs when attempting to read a file but cannot access the file. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error accessing file: {0}
OnExecRunInvalidParameterFile	Occurs when attempting to read the parameter file but cannot access the file. For example, the file does not exist or the parameter file URI is invalid. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnExecRunInvalidTestReport	Not Implemented

OnExecRunInvalidTestbed	Occurs when the testbed is invalid. For example: the testbed file does not exist, the testbed file cannot be loaded, the testbed URI is not valid, and so on. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnExecRunInvalidTestcaseUri	Occurs when the test case is invalid. For example: the test case does not exist, the test case cannot be loaded, the test case URI is not valid, and so on. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}
OnExecRunResultMatch	Occurs when the running result matches the expectation. **Default actions:** • **PassTestIfNotAlreadyFailed** • **DeclareExecutionIssue** with a default **Message** of: OK: Test case "{testcase}" run result was "{completion}" as expected.
OnExecRunResultMismatch	Occurs when the running result does not match the expectation **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Unexpected result "{completion}" when executing test case "{testcase}". Expected "{expected}"
OnExecSummarizeError	Occurs when generating the running result summary has exceptions **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Fail to generate summary : {0}
OnExecSyntaxError	Occurs when executing a step with invalid syntax. For example, a **for** statement with fewer than three arguments or the **sleep** time is not a valid number **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid syntax: {0}
OnExecUnknownActionError	Occurs when executing a step with invalid action that cannot be interpreted by the iTest kernel **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Unknown action: {action}
OnExecWriteError	Occurs when executing a **write** action that is not in the current thread **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Write action is not supported in a new thread outside its own procedure context. Write will be ignored.

OnExecWriteFileError	Occurs when exceptions occur while attempting to write a file. For example, the file URI is not valid or the file cannot be written **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error writing to file: {0}
OnExecWriteFileSyntaxError	Occurs when executing a step with invalid write file syntax. For example, requires 2 arguments but only 1 is provided. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error writing to file: Syntax Error ({0})
Execution:Tcl Interpreter	
OnTclInterpreterError	Occurs when Tcl fail to interpret actions such as set, get, or eval **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error using Tcl interpreter: {0}
OnTclInterpreterUnavailable	Occurs when the session executor tries to get the Tcl interpreter but failed. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: TCL interpreter unavailable: {0}
Processor	
OnUnknownProcessorError	Occurs when executing a step with invalid write file syntax. For example when two arguments are required, but only one argument is provided. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error writing to file> Syntax Error ({0})
Response Mapping	
OnApplicableMapsNotFound	Occurs when no applicable response map could be used during step post-processing No default actions
OnMappingFail	Occurs when response mapping fails during step post-processing. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {message}
OnMappingInfo	Occurs when response mapping passes. Provides standard information about the mapping process during step post-processing. **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Information** and with a default **Message** of: Information:{0}

OnMappingResponseEmpty	Occurs during step post-processing if the response is empty. No default actions
OnMappingWarning	Occurs during step post-processing.when response mapping passes but with warning information about the mapping **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: {0}
OnResponseMapLibraryNotFound	Occurs when the response map library could not be found during step post-processing. For example, the map library path is not valid. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {message}
OnResponseMapNotFound	Occurs when the response map is not available during step post-processing **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {message}
OnResponseMappingIssue	Occurs for each response mapping issue during step post-processing **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {message}
OnResponseMappingWarning	Occurs for each “warning” level response mapping issue during step post-processing **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: {message}

Session Handling

OnOpenWithSessionAlreadyActive	Occurs when error happens when trying to execute an **open** step in an active session **Default actions:** • **AbortStep** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Session "{session}" is already opened and therefore action "{action}" is not allowed
OnSessionDeviceNotFound	Occurs when session device or session profile is not found. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Device "{0}" was not found in the testbed

OnSessionIllegalAction	Occurs when session is in illegal state **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Action "{action}" is not allowed when the session is in the current state. Step will be ignored.
OnSessionNoTestbedAvailable	Occurs when the device URI cannot be used because there is no testbed available. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: A device URI cannot be used because there is no testbed available
OnSessionNotOpen	Occurs while attempting an action if the session is not open **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Session "{session}" is not open and therefore action "{action}" is not allowed
OnSessionOpenFileMissing	Occurs when the file referenced in open step is missing. **Default actions:** • **AbortStep** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: File referenced in open step "{file}" is missing
OnSessionOpenInvalidProfile	Occurs when there are errors or warnings when reading session profile. **Default actions:** • **AbortStep** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Errors or warnings when reading session profile
OnSessionOpenSyntaxError	Occurs when session open syntax is invalid. **Default actions:** • **AbortStep** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Invalid syntax on open: {0}
OnSessionProfileNotFound	Occurs when session profile is not found. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Session profile for device "{0}" was not found in the testbed

OnSessionStepAfterSessionTerminated	Occurs when executing session steps after the session has terminated **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Session has already terminated. Remaining steps in this session will be skipped.
OnSessionStepExecutorCreateFail	Occurs when errors happen while creating session step executor. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error creating step executor
OnSessionStepSubstitutionError	Occurs when command and variable substitution failed on the step. **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: Command and variable substitution failed on the step: {0}
OnSessionUnknownApplication	Occurs when the application for the application name is unknown **Default actions:** • **AbortStep** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: The application "{appName}" is unknown
OnUnsupportedSessionAction	Occurs when the action is not supported by this type of session. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Action "{action}" is not supported for this type of session

Terminal-Based Applications

OnProcessorNotFound	Occurs when executing the command step in HA mode but the processor can't be found. **Default actions:** • DeclareExecutionIssue with a Severity of Error and with a default Message of: Error: Unable to find a processor in appropriate state: {0}
OnPromptNotFoundTimeout	Occurs when executing a step but no prompt found and the wait time is exceeded. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Step timed out waiting for the prompt
OnTclCreateInterpreterFail	Occurs when creating local TCL interpreter fails. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: {0}

OnTerminalConnClosedBeforeCommand	Occurs when the terminal connection has been lost before sending the command. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Connection was already closed before sending the command "{0}"
OnTerminalConnClosedWaitingForResponse	Occurs when waiting for the step response but the terminal connection has been closed. No default actions
OnTerminalLoggingFailure	Occurs when logging the terminal session to file has IO exceptions **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Unable to log session data: {0}
OnTerminalOpenConnectionFail	Occurs when opening the command session has exceptions. For example, the session parameter is invalid or the session has already opened **Default actions:** • **AbortExecution** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error: Error attempting to open connection to server: {0}
OnWaitingForPromptTimeoutExtended	Occurs when waiting for the prompt in terminal session exceeds the max idle time. And indicate the user to increase the "Extra wait before alerting" property **Default actions:** • **DeclareExecutionIssue** with a **Severity** of **Warning** and with a default **Message** of: Warning: You waited for a prompt. Consider increasing the \"Extra wait before alerting\" property to {0} seconds.

Tool

OnToolCloseSessionError	Occurs for the following cases: • No **open** action is defined before **close** action • Unexpected **corrupted** state • Try to execute close action when the session is in NOT_STARTED or OPENING state **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Error closing session : {error}
OnToolCreateSessionError	Occurs for the following cases: • Session opening was interrupted • Underlying Eclipse throws internal error when opening session **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Unable to create session : {error}

OnToolError	Occurs when a test case step is executed before corresponding session is opened. **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: {error}
OnToolOpenSessionError	Occurs for the following cases: • Session is already open • Session is in the process of being closed by another thread • Session is in the process of being opened by another thread • Open method is not defined for this kind of profile or testbed or topology **Default actions:** • **FailTest** • **DeclareExecutionIssue** with a **Severity** of **Error** and with a default **Message** of: Unable to open session : {error}

Deferred actions

Because some actions alter the flow of execution, they are deferred (not executed) until all other actions for the step are executed. This is true even if the action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after an **AbortExecution** action, the **AbortExecution** action does not execute until the **FailTest** action is finished executing. The following actions are deferred and are fully described in the table in “Actions on events: Definitions” on page 626:

AbortExecution

Break

CallProcedure

Continue

ExitExecution

ExitProcedure

Goto [Deprecated. **We strongly recommend that you not use this action**.]

Actions on events: Definitions

Events (described in “Event definitions” on page 606) can perform the following actions:

AbortExecution	Stops execution. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after an **AbortExecution** action, the **AbortExecution** action does not execute until the **FailTest** action is finished executing. No configurable properties.
AbortStep	Immediately stops executing the step and then proceed to the next step. In an analysis rule, this action has the effect of not performing remaining analysis rules and then continuing to the next step. No configurable properties.
AbortTest	Set the test result as **Abort**. Continue to execute. Note that execution is not stopped. No configurable properties.
AbortThread	Immediately stop executing the current thread. No configurable properties.
AppendToProcedureResponse	Writes the specified text to a procedure’s return buffer. To return the full response, set the **Content** text to **[response .]** Supports field replacements.
AppendToStepResponse	Writes to the specified text to a step’s return buffer. To return the full response, set the **Content** text to **[response .]** Supports field replacements.
Break	Break out of a **for**, **foreach**, or **while** loop. Use this **break** action to stop executing a loop and continue executing at the step after the loop. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **Break** action, the **Break** action does not execute until the **FailTest** action is finished executing. No configurable properties.

CallProcedure	Call the specified procedure (local or foreign) or the specified QuickCall. iTest provides interactive support for specifying the procedure. See “Calling a procedure in a test case step or in a property setting” on page 263. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **CallProcedure** action, the **CallProcedure** action does not execute until the **FailTest** action is finished executing. **Note** Do not add a **CallProcedure** action to either an **OnProcedureEnter** or an **OnProcedureExit** event—this results in an infinite loop. **Property:** **Procedure name and arguments**: Specify the procedure name followed by argument values if appropriate. For example, **SetupDevice -deviceNumber 3**.
Continue	The **Continue** action causes the current step to be aborted out to the innermost containing for, foreach, or while loop command. The loop then continues with the next iteration of the loop. Use the **Continue** action when you want to execute particular steps for some iterations of the loop, but not for other iterations. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event and if there are additional analysis rules listed after the current rule for the step. For example, even if a **FailTest** action appears last in the list after a **Continue** action, the **Continue** action does not execute until the **FailTest** action is finished executing. No configurable properties.

DeclareExecutionIssue	Generate an execution issue with a particular severity and display an associated execution message in the Execution view, in the Step Issues view, and in test reports. **Properties**: **Severity**: This setting specifies the type of issue: OK (executed as expected), Information, Warning, Error (did not execute as expected) **Note** In HTML, text, and XML format reports, the **OK** severity is listed as “pass” and **Error** is listed as “fail”. **Message**: Specify the text message to display in the Execution view, in the Step Issues view, and in test reports. Field replacements are supported. **Note** iTest can generate a plain language sentence for the execution message (for example, “Extracted value $value is equal to “Up”). To use this feature, specify a **Message** value of **{auto_message_true}** or **{auto_message_false}**, as appropriate. You can use any of several built-in variables to customize the message text. Information on built-in variables like **$value** appears in “Tcl interpreter local variables” on page 340. **Note** To ensure access to certain data that is available when the message is generated, iTest first applies its standard field substitution and then uses Java-style format strings for messages. Java format strings uses escaping rules that differ from Tcl rules. For example, Java string format uses single quote ' as its special character and you need two of these for escaping. So, to cause ‘ to appear in the message, use two single quotes ‘’ in the message text. For the Java string format rules, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/MessageFormat.html
Eval	Evaluate the statements specified in the **Properties** cell. No configurable properties.
ExitExecution	Immediately stop executing the current procedure and all threads to end test case execution. The **ExitExecution** action does not change the existing execution result of the test case (Pass, Fail, Abort, or Indeterminate). **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **Break** action, the **Break** action does not execute until the **FailTest** action is finished executing. No configurable properties.

ExitProcedure	Stop executing the current procedure and return execution from the current procedure to the caller. You can specify a return value for the procedure in the Return value property. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event and if there are additional analysis rules listed after the current rule for the step. For example, even if a **FailTest** action appears last in the list after an **ExitProcedure** action, the **ExitProcedure** action does not execute until the **FailTest** action is finished executing. No configurable properties.
FailTest	Set the test result as **Fail**. Continue to execute. No configurable properties.
PassTest	Set the test result as **Pass**. Continue to execute. No configurable properties.
PassTestIfNotAlreadyFailed	If the test result is not already **Fail**, then set the test result as **Pass**. Continue to execute. No configurable properties.
PauseExecution	Immediately pause execution. Bring the Execution view to the front. No configurable properties.
RepeatStep	Repeat the current step. The **RepeatStep** action cannot be used with an async step (a step that has the **Start this step (in a new thread) and proceed to the next step** property checked). **Properties**: **Maximum repeat count**: Specify the maximum number of times to repeat the step. **Delay between repeats**: Specify how long to wait between repetitions of the step (in seconds).
ScriptEval	Evaluate the script specified in the **Properties** cell using the Tcl interpreter attached to the execution kernel. The response is populated (including structured data) in a way consistent with how a Tcl Shell Command step works (including result, STDOUT, and/or STDERR). The following types of substitution are not made to the text of the **Command** property before the step is executed: • Command field replacements • Variables • Backslash characters used to escape special characters No configurable properties.

SetResponseValue	Configure a action response, which returns the appropriate value during test execution. For example, to specify the actions that should occur upon True and False results, specify Rule Actions for **When True** and **When False**. That is, enter the Action**SetResponseValue** and specify a response. See also “Assert processor” on page 651 (Chapter 29, “Analysis Rules: Validating Responses”).
SkipRemainingRules	iTest does not perform any further analysis rules associated with the current step. For example, you might set this Action so that when an analysis rule concludes that something has gone wrong and there is no point in performing additional analysis, then skip further analysis. No configurable properties.
signal *eventName*	For synchronous execution: Wakes a thread that is waiting on *eventName* and causes it to continue execution. See “signal: Wake a thread that is waiting on an event” on page 594.
signalActivate *eventName*	For synchronous execution: Turns on the event called *eventName*. See “signalActivate: Turn a signal on” on page 595.
signalAll *eventName*	For synchronous execution: Causes the currently executing thread to sleep until all specified events have been signaled or activated (**signal**, **signalAll**, or **signalActivate**). See “signalAll: Wake all threads that are waiting on an event” on page 594.
signalClear *eventName*	For synchronous execution: Removes any instances of the event named *eventName* that had previously been activated either by a **signalActivate** step or by a **signal** command. See “signalClear: Deactivate a signal” on page 595.

CHAPTER 28

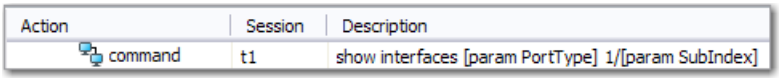
Field Replacements

Field replacements: Substituting values into properties and commands

Just before executing a step, iTest evaluates *field replacement* text and replaces the text with the values of a variable or a parameter or a value returned in a response. This process, often called *runtime substitution*, is available in commands, property settings, or messages — most text fields.

Example: Using an iTest Tcl interpreter 'param' command in a field replacement

1 The text strings **[param PortType]** and **[param SubIndex]** appear in the following step:



The **[param PortType]** syntax means "Before running this step, replace all of the text in this field with the value of the **PortType** parameter". (We will describe the syntax in a moment.)

2 The test case has the following parameter settings:

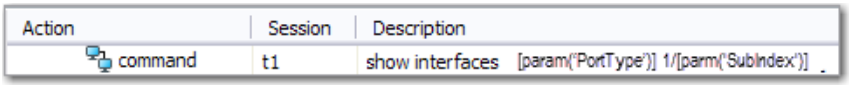
```
PortType = FASTETHERNET
SubIndex = 0
```

3 At runtime, step preprocessing replaces the fields. As a result, iTest issues the following command:

```
show interfaces FASTETHERNET 1/0indicator
```

Example: Using an iTest Python interpreter command in a field replacement

1 The text strings **[param('PortType')]** and **[param ('SubIndex')]** appear in the following step:



The **[param ('PortType')]** syntax means "Before running this step, replace all of the text in this field with the value of the **PortType** parameter"

2 At runtime, step preprocessing replaces the fields. As a result, iTest issues the following command displaying the type of values, ins:

```
PortType = FASTETHERNET
SubIndex = 0
```

3 At runtime, step preprocessing replaces the fields. As a result, iTest issues the following command:

```
show interfaces FASTETHERNET 1/0indicator
```

How field replacements operate

At runtime, before the property or step is interpreted, iTest substitutes the returned value in place of the field replacement text. Field replacements are done in context of the testcase.

Here is how that works in Tcl:

Each executable step is ***preprocessed*** immediately before execution. Preprocessing means that all user-defined text associated with the command is searched for any field that starts with **[** and ends with **]** (this is, iTest searches for field replacements). The entire field, including the brackets, is replaced with a string determined by evaluating the information inside the braces. (User-defined text includes prompts, the command body text — which may be multi-line for some protocols — the command header text, and the user-defined text values in the associated Analysis rules. This means that you can use a field replacement in any of these locations.)

If the operation inside the **[ ]** braces results in an error, then iTest generates an execution issue and displays a message in the Execution view, the Step Issues view, and in the test report. The text of the field replacement is not substituted for execution.

In the Tcl example, the **param** command returned the value of the **PortType** parameter, so that is the value that replaces the **[param PortType]** text in the **show interfaces** command.

Here is how it works in Python

Fields with substitution/replacements enabled (for example, the command field for a command action in SSH) will use braces, [ ] as escape characters for interpreter evaluation. iTest evaluates everything between the braces, [and] characters, using with the Python interpreter, and replaces with a string determined by evaluating the information inside the braces.

Note It is not necessary to escape [and] characters within the braces.

In the example, **show interfaces [param('PortType')] 1/[param('SubIndex')]** would evaluate PortType and SubIndex defined parameters.

The param command returns the value of the **PortType** parameter, the value that replaces **[param('PortType')]** in the **show interfaces** command.

Example 2: ping -c 3 [hostList[0]]

The **ping -c 3 [hostList[0]]** command would evaluate to ping the first hostname in a Python list called **hostList**.

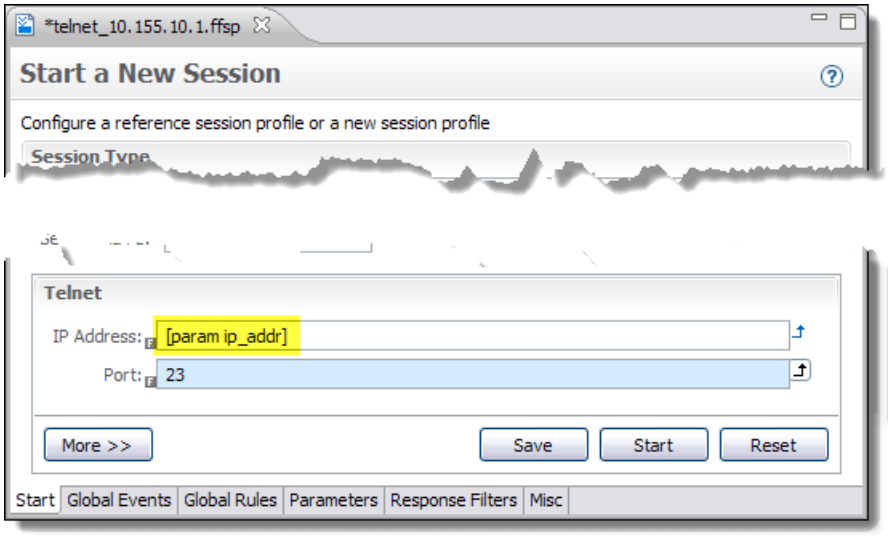
Example 3: Field replacement obrains value required by the command from the test case execution context. That is, to obtain a user mail address in Mail (SMTP) session, use command as follows: [**profile('.' ,'Mail/user')]** to obtain value from the user profile (not **[param('mail/user')]**, which will obtain value from parameters).

Alternatively you may use **[param('.//profile/Mail/user')]** to allow xpath to obtain the parameter value.

Where you can use field replacements

You can use field replacements in most user-defined text or numeric values that are used during execution, for example:

- Test case step properties like **Session**
- The text of test case **Comment** steps
- User-defined text values in analysis rules, for example, the **Extractors** set of properties
- Session profile properties like the IP address of the device. For example:



Note The [param], [get], [gget] should have default values, if not, the variable and parameter substitution will fail when a session profile is manually started.

- The text of the **message** property in events and analysis rules
- Prompt **Content** property values (that is, the text that appears in the prompt)
- test case **command** and **variable** substitution. See the following examples:

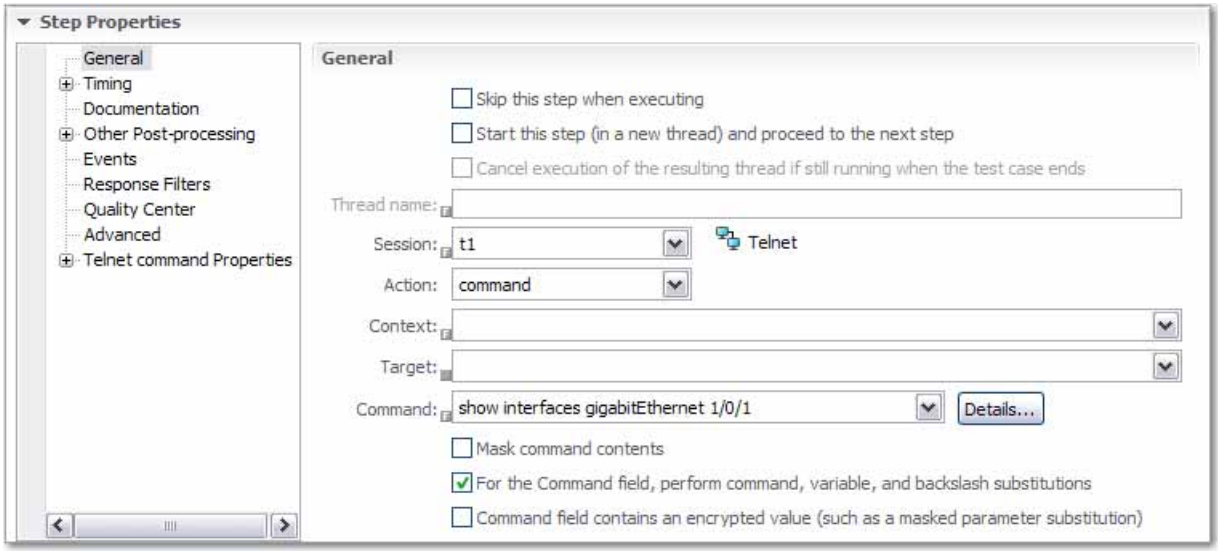
  Example: **[iTestVersion]** might return the float value of 7.1, **[iTestPlatform]** might return the string **'eclipse'**, and **[f.readline()]** might return the string **"next line of file text"**

  Example: command echo **[import sys; sys.platform]**

Note Python uses the Square brackets to determine expression only for session commands.

- Supports '['
- Example: "\[qwe]" will be passed as [qwe] without substitutions.

iTest displays an F icon on the text box for properties that will undergo substitution of field replacement text at runtime. In the example, you can see that the **Thread name**, **Session**, **Context**, and **Command** properties will be substituted at runtime if the property string includes field replacement text. Notice that the ■ indicates that iTest will not perform substitution for the **Target** text.



Important The **Command** property is special in that it can also perform variable and backslash substitution and a command might be multiline for some protocols. To allow you to specify the behavior exactly as you need it, iTest provides the **For the Command field, perform command, variable, and backslash substitutions** check box. See "Step Properties section: General properties group" on page 176 for details.

Enabling and disabling runtime substitution of field replacements

To specify whether or not to perform substitution of field replacement text before interpreting the text at runtime, click the F (the field replacement indicator). The indicator switches among one of the following states:

F	Perform substitution of all field replacements before interpreting the text
■	Do not perform substitution of text that has the form of a field replacement
?	This indicator appears if you select multiple steps in the Test Case editor and the steps have different settings.

To specify the substitution setting for multiple steps, select all of the steps (Ctrl-click and Shift-click) and then click the F indicator.

Important If the field replacement text includes a character that should not be replaced at runtime, be sure to escape it using the backslash **** character. For example, **\$** escapes the **$** character.

Property inheritance

For a property that inherits its value, you must turn off inheritance before changing the substitution/no substitution setting (because the setting is also inherited).

In this example, we want to change the **Connect timeout** value from its inherited value of **30** to a **param** command whose value is determined at runtime (as a result, depending on a parameter value, the test case can wait 45 seconds to connect to **DeviceA** and 120 seconds to connect to **DeviceB**, for example).

Before we can change the ■ (do not substitute) setting to the ▣ (substitute) setting, we must first click the inheritance button to turn off inheritance. (For more information on how properties inherit their settings, see "Property values: Inheriting settings" on page 62.)

Field Replacement Tip: Use the Insert Field tool

While inserting a field replacement, you do not have to type out a field replacement or its command and argument. Just right-click anywhere that you can add a field replacement and select **Insert** to insert a properly formatted field replacement with hints about argument usage. See "Inserting field replacements using the Insert Field tool" on page 637 for details.

General format of field replacements

Tcl

In Tcl, the generic format for defining a field replacement is:

[commandName args] or commandName('args') in Python

for example, here is the **param** command in a field replacement:

Tcl: [param portCount]

Python: [param('portCount')]

Some commands require a single argument, others use a subcommand and multiple arguments. In the example, **portCount** is the single argument. Subcommands and arguments are separated by spaces.

iTest interpreter commands are fully described in Chapter 17, "iTest Commands".

Commands that are often used in field replacements

Use the **Insert Field** tool as described in "Inserting field replacements using the Insert Field tool" on page 637 to view the full list of supported field replacement commands.

Here are some commands that are commonly used in field replacements:

char: See "char command: Inserting non-printing characters" on page 371

expr: See "expr command: Evaluating expressions" on page 372

param: See "param command: Returning parameter values" on page 372

profile: See "profile command: Accessing parameters that are defined in session profile" on page 373

query: See "query command: Inserting the results of a query" on page 374

response: See "response command: Accessing response data that is stored in a variable" on page 376

tcl: See "tcl command: Evaluating Tcl statements in the execution kernel's Tcl interpreter" on page 377

tclexpr: See "tclexpr command: Evaluating Tcl expressions in the execution kernel's Tcl interpreter" on page 378

call: See "call command: Call a procedure and get the return value from that procedure" on page 378

json_select: see "jsonSelect command: Get the node value from json string based on the query xpath" on page 380

Guidelines for using field replacements

- If the first separator **[** is present, but the second separator **]** is not, then iTest generates an error.
- To include the character **[** as text within a command (that is, not as the initial characters of a field replacement), type **\[**
- Whenever a single one of the characters **** or **[** appears in text, it results in an error and an Execution Message is displayed in the Execution view and in the test report.
- Field replacements can appear anywhere in the text.
- Field replacements support nesting. The innermost replacement is processed first, followed in succession by each more outward replacement. Use the syntax shown here:

  [command_2 [command_1:value_1] argument_2]

  For example:

  [param [param portIndex] State]

Variable substitutions

Variable substitutions are performed as follows:

Tcl	Python
On any string Starting with $ (for example, $j)	Any strring: **j** (non-session commands) and **[j]** (session commands)
For variable names with spaces, use **${variable name}**	Does not support spaces in variable names.
**port${i}count**: If the variable **i** has value 10, then this is replaced by port10count **Note** QC session constant in Tcl: "$session"	For variable **namesport[i]count**. If the variable **i** has value 10, then this is replaced by nameport10count **Note** QC session constant in Python: "[session]"
The **$** character is escaped by a single backslash ****	Backslash is not required in Python.

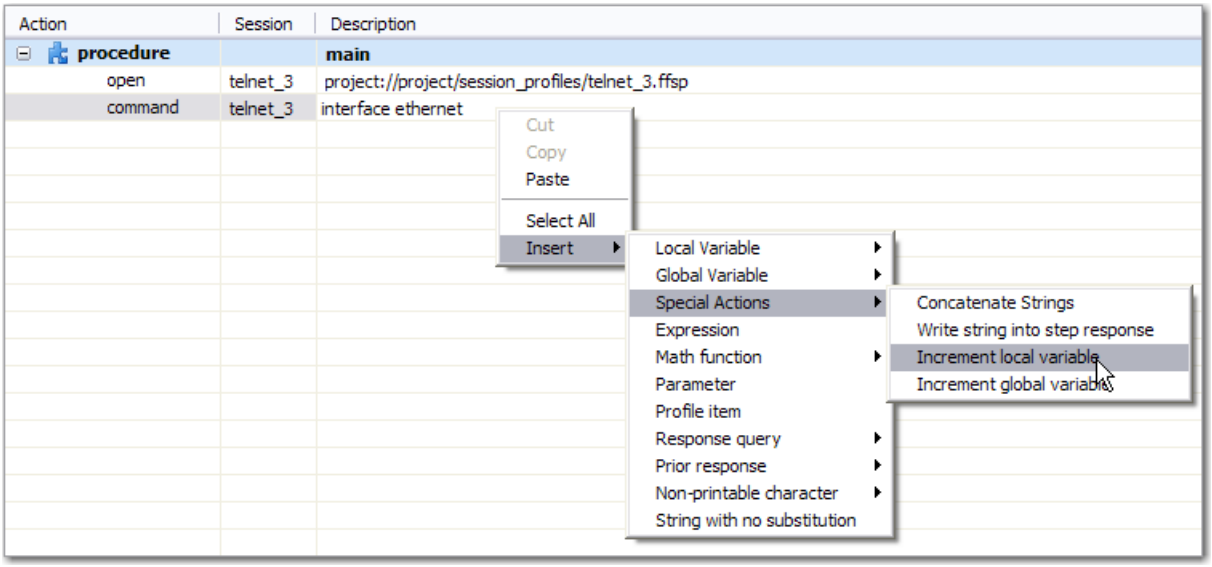
iTest includes some useful pre-defined Tcl variables. See "Tcl interpreter local variables" on page 340.

Inserting field replacements using the Insert Field tool

While editing a step, you can right-click to open the **Insert Field** tool to help you to build and insert a field replacement. In this example, we'll use the tool to complete the **Interface ethernet** command by adding a field replacement that is replaced by the value of the **Port** parameter.

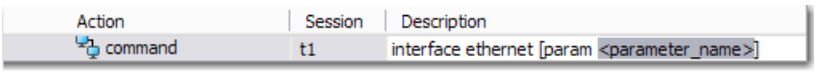
Using the Insert Field tool

Place the cursor where you want to place the field replacement text (in the example, after the word **ethernet** in the command). Right-click and, from the menu, select the type of command field replacement to add. The tool displays appropriate choices until you reach a final choice.

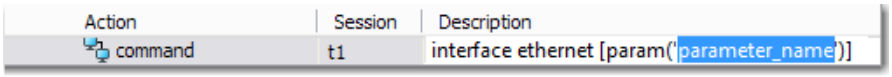


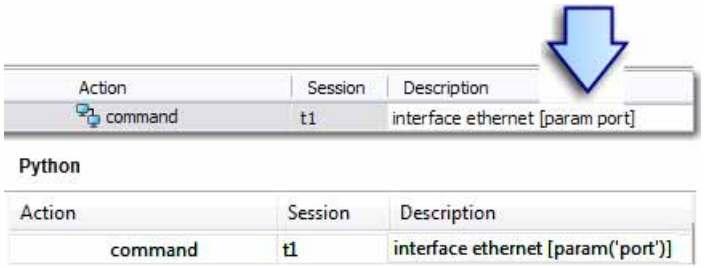
Select the value to paste the field replacement text into the cell at the cursor position. In the example, we insert a parameter. The correct field replacement text is inserted. It remains only for us to replace the **<parameter_name>** placeholder text with the name of a parameter defined

for the test case (**port** in this example). As a result, the **[param port]** field replacement is appended to the command text.



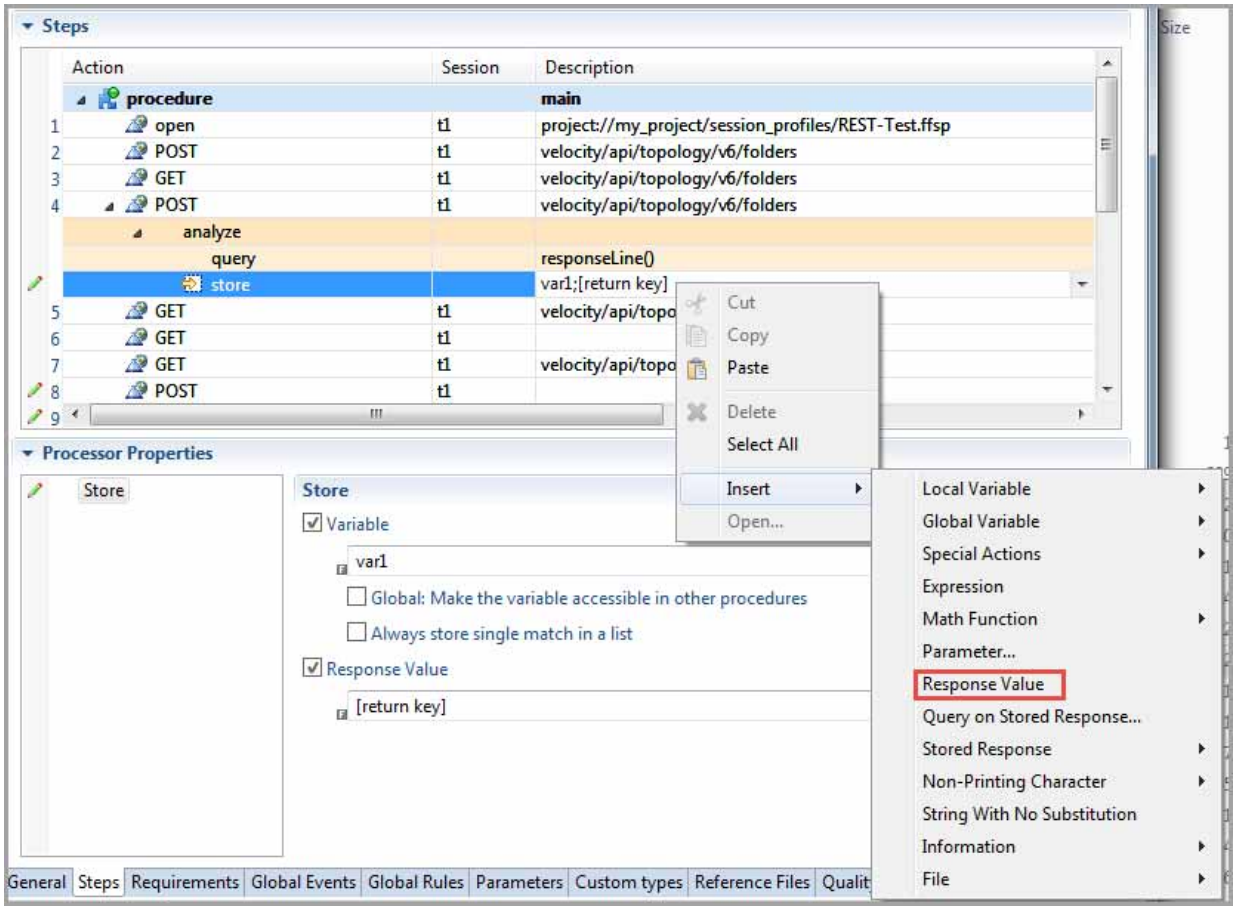
Python:





Insert Response Value for JSON Response

The Response Value sub menu is only available for steps of a Procedure when **JSON Resonse** is enabled (see page 223).



Where you can use the Insert Field tool

Editable text boxes for properties that support field replacements support the Insert Field tool. Right-click in a box to test. For example, the session and Description cells in the Test Case editor and most step property fields.

Supported field replacement types

The Insert Field tool supports the field replacement types shown in the screenshot.

Note The contents of the **Insert > Query on Stored Response** option are taken from the most recent test report, so, to populate the options in the list, you must execute the test case at least once before using **Insert > Query on Stored Response**.

Storing response data into a variable

See "Storing a response into a variable (for use later in the test)" on page 140.

CHAPTER 29

Analysis Rules: Validating Responses

Analysis rules: Validating responses and setting Pass / Fail

One of the most important functions of a test case is to analyze (validate) the responses to steps, and then, based on the results of the analysis, take some specified action like set the test result to **Pass** or **Fail**, and add an execution message to the test report about why the test passed or failed, for example, “Procedure **clearStats** succeeded in clearing the stats” or “Configuring Router2 failed at port config step”.

In Spirent iTest, analysis rules specify both how to analyze a response and the actions to take based on the analysis of the response.

Adding an analysis rule

The easiest way to add an analysis rule to a test case step is:

1. Select the step in the Test Cas**e** editor
2. In the Response view, right-click in the appropriate blue box (that is, the data to validate)
3. Select **Analysis Rule Wizard**

About analysis rules

- Analysis rules are optional. You create a rule only when you need to work with the response to a step. Typically, you define rules for only a few steps.
- Each executable step can have any number of analysis rules associated with it. Rules are applied in the order in which they are listed.
- An analysis rule applies only to the step for which it is defined. (You can define special Global rules)
- During execution, if the condition that you specify in a rule is not met, the default action is to display an execution issue in the Execution view and in the test report. You can specify additional actions (like passing or failing the test and so on).

Examples of the types of Pass / Fail validation that Analysis rules can perform

- The response must include the text value “restarted”
- Perform the specified action if the specified expression evaluates to False.
- Perform the specified action if the specified regular expression evaluates to True.
- The response must appear within 3 seconds of sending the command
- The **DroppedPackets** value in the response must equal zero.

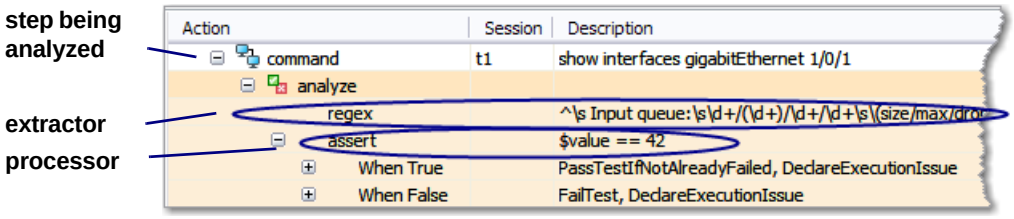
- The **InstalledCards** value in the response must lie between 4 and 6.
- Perform the action specified for the rule regardless of the response.

Examples of the types of action that Analysis rules can perform

- Cause test case execution to continue at a particular place (for example, a procedure that reconfigures or initializes the DUT).
- Pass the test case and place the message “Ports successfully configured” in the test report and the Execution view.
- Execute a Tcl script if a particular value is greater than expected.

The structure of an analysis rule

Analysis rules have two main parts; the **extractor** and the **processor**:



The extractor

The extractor is the first line in the rule — it specifies ***how to extract (return) a value*** from the response and ***what to extract***.

- ***How to extract the value*** appears in the **Action** cell. In the example, we use a **regex** (regular expression) extractor to extract the value. Alternatively, you can use the **query** extractor or the **contains** extractor that extracts the text string that you specify. Some extractor types can return multiple values.
- ***What to extract*** appears in the **Description** cell. In the example, it is the regular expression that defines the text to extract from the response.

Limitations

The following limitations apply for the data extracted for each execution:

- **Total elements stored:** A maximum of 128 extracted data items per execution.
- **Bytes stored:** A maximum of 128 characters of any tag or value. Any tag or value that exceeds 128 characters will be truncated.
- **Array elements stored:** Any extracted data item whose value is an array that exceeds 128 items will be rejected (discarded).

The processor

The **processor** is the second line in the rule — it specifies ***the type of action to take*** and ***the details of the action to take on the extracted data***.

- ***The type of action to take*** appears in the **Action** cell. In the example, the action is to use the **assert** processor to test an assertion about the extracted value. Other processors **chart** the value, display an execution **message**, or **store** the value in a variable.

- ***The details of the action to take*** appear in the **Description** cell. In the example, we specify the assertion to be tested: test whether the extracted data equals **42**. The **When True** and **When False** substeps are a part of the **assert** processor that tell iTest to take a particular action when the assertion is true and a different action when the assertion is false.

  If we had specified a **message** processor, the details of the action would be the text to display in the message.

Adding and working with analysis rules

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

You can build an analysis rule by hand, entering the setting for each property. The easiest way to add an analysis rule, however, is to select the step in the Test Case editor, right-click on the appropriate key value in the Response view, and then select Analysis Rule Wizard. See "About the Analysis Rule wizard" on page 667 for details.

Limitation

For an analysis rule that stores a match as a global variable: If you breakpoint the step immediately after the step that contains the analysis rule, then the value will not be stored in the global variable.

Using the Analysis Rule wizard to specify an analysis rule for a step

The wizard walks you through the process of creating the most common rule types.

Follow these steps:

1 In the Test Case editor, select the step whose response includes the data to validate.

Note You can add a rule only if the step is selected in a test case as viewed in the Test Case editor. You cannot add a rule if the step is selected in a test report as viewed in the Test Report editor.

---

2 Right-click any one of the following items and then select **Analysis Rule Wizard**.

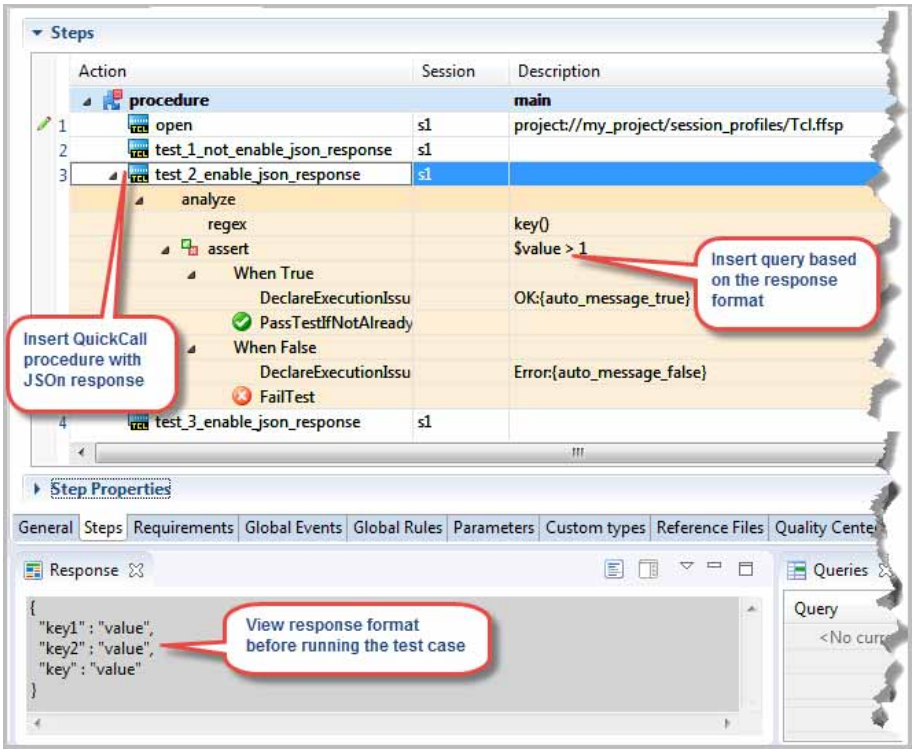
- Right-click the step in the Test Case editor
- In the Response view, right-click the data to validate
- In the Structure view, right-click the item that returns the data
- In the Queries view, right-click the query that returns the data

3 The **Analysis Rule** wizard starts. Follow the instructions that appear on the wizard pages.

Advanced usage tip—streamlined Analysis rules

You may define Analysis Rule before executing a test case by defining QuickCall procedures or custom session with JSON Response, and inserting these calls in a Test Case. The insterted step populates the **Response View** with the JSON response format from the called procedure

(the Response View background is light grey before running a test case). You may insert Analysis Rule as required before executing the test case. See the example illustration below.

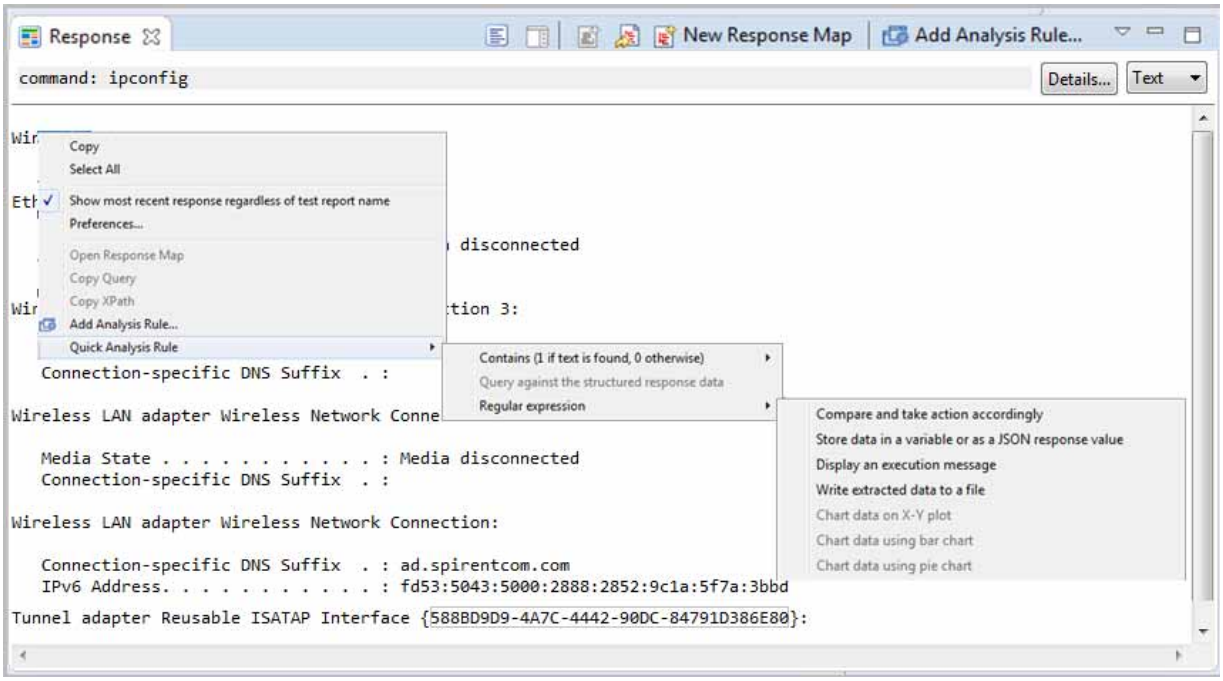


Using the Quick Analysis Rule feature to specify an analysis rule for a step

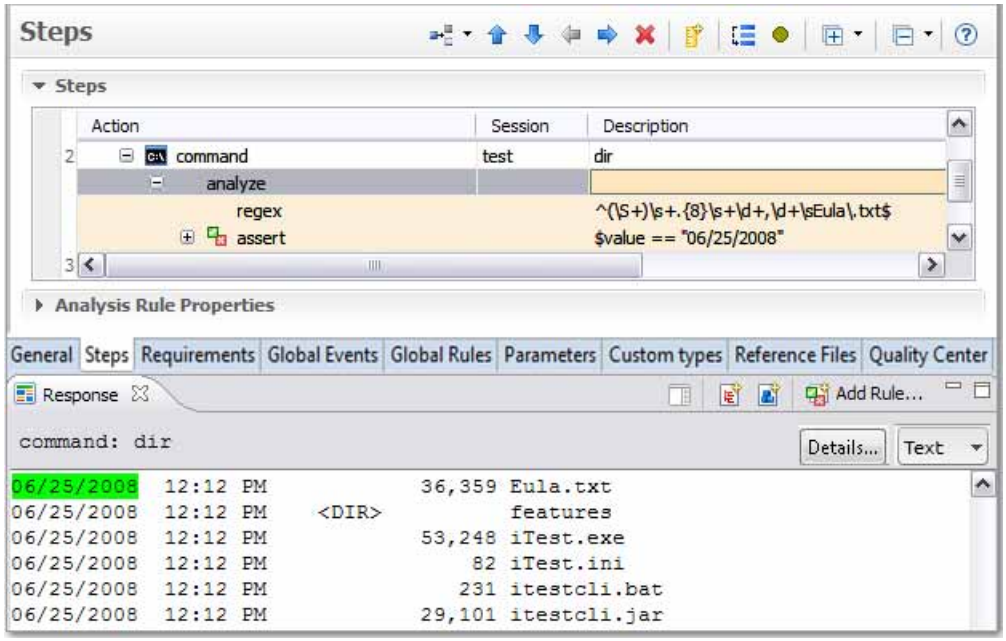
In this example, we'll add a regex rule that checks whether a value is correct and then takes action based on the result.

1 In the **Steps** section of the test case editor, select the step. The response appears in the Response view:

2 iTest places the response to the step (from the last execution) into the Response view. Select the text to analyze. Right-click the selection and then choose **Quick Analysis Rule > Regular expression > Compare and take action accordingly**.



A new rule is added to the step, specifying the regular expression as the extractor, and **assert** as the processor, This rule will place the extracted data into the assertion listed in the Details property and then test the assertion. Notice that the item is highlighted in green in the Response view (the assertion returns 1 (True)).



3 Now, check whether the rule works as you expect. Open the Step Issues view to see the execution message that results for the response text from the most recent execution of the

step (that is, the response text in the Response view that you used to help you to create the rule). This method saves you from having to iteratively execute the test case to see whether the rule works correctly.

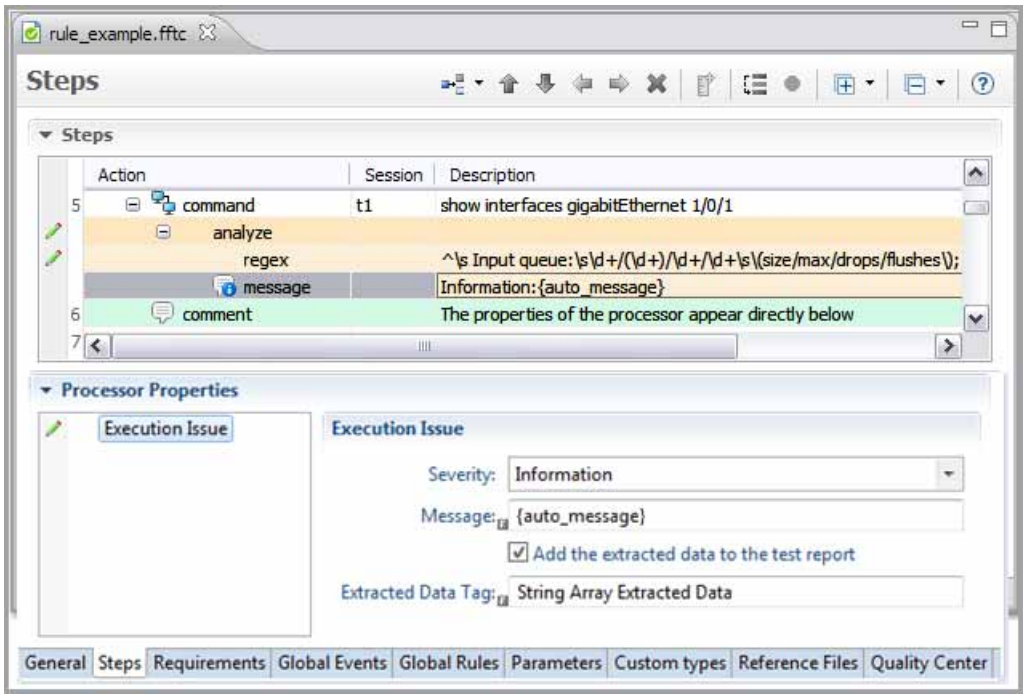
Editing Analysis Rules

1. To make changes to an analysis rule, first select the appropriate line in the rule.
2. Select the field that you want to edit. Now, either edit the text directly or specify new property settings for the rule.
3. To modify a property setting, open the **Extractor Properties** section or **Processor Properties** section as appropriate.

Note You may open the context specific information—**Processor Properties** or **Extractor Properties** section in the **Properties** pane as follows.

- Right-click to display the menu and select **Show Properties View**.
- Click the ellipsis on the step command (where applicable).

You may edit properties using either the **Processor Properties** section (within the Test Case Editor) or via the **Properties View** tab.



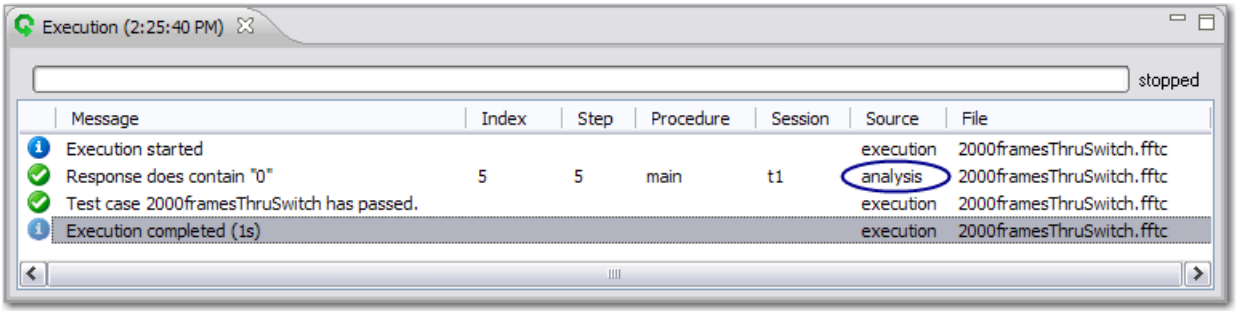
Deleting Analysis Rules

To delete an analysis rule, select it in the steps grid and click  .

Testing Analysis Rules

Verify that your analysis rules are performing properly by executing the test case and reviewing the results in the Execution view (pass/fail result as expected, messages output properly, and so

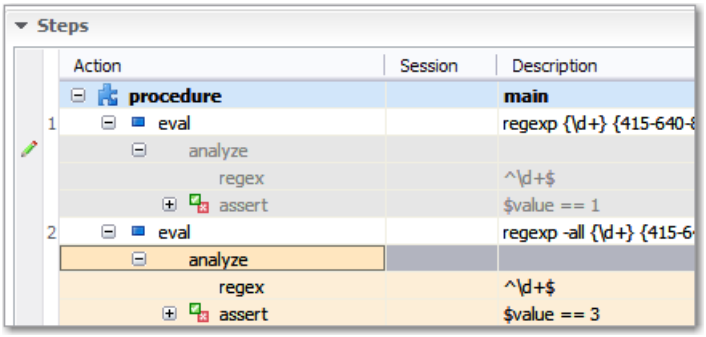
on) or the Step Issues view. Notice that the analysis rule is cited as the reason for the execution message with a **Severity** of **OK** .



Skipping Analysis Rules

While developing and troubleshooting a test case, it is often helpful to not analyze a particular response, that is, to skip execution of an analysis rule (for example, a rule that you know will fail while you are developing the test case).

- Skipped analysis rules are not executed and do not appear in reports
- In the steps grid, skipped rules are dimmed (grayed-out). In the example, the rule for step 1 is skipped and the rule for step 2 is not skipped.



Skipping and unskipping analysis rules

1. In the Test Case editor, select the analysis rules (use Ctrl+click or Shift+click for multi-select).
2. Click **Skip** . (Alternatively, for a single selected rule, in the **Analysis Rule** property page, check the **Skip this analysis rule when executing** check box.)

Click the button again to unskip rules.

Skipping and unskipping Global analysis rules

On the Test Case editor's **Global Rules** page, the **Skip** check box indicates whether a rule is skipped. Use either of the following methods:

- Select the rules (use Ctrl+click or Shift+click for multi-select) and then click **Skip** in the toolbar. The **Skip** check box is checked for the rules.

  or

- For any rule, check the **Skip** check box.

Click the button again or clear the check box to unskip rules. In the example, the **regex** rule is skipped and the **contains** rule is not skipped.

Analysis rules: Properties of the extractor

This topic describes the properties for each type of extractor.

- In the Test Case editor, the processor is the first line in an analysis rule.
- For Global rules, the processor is displayed in the **Extract using** and **What to extract** cells.
- In the Response Map wizard, the **Extractor** page offers the option to specify the properties described in this topic.

Predefined local variables used by extractors

iTest populates predefined local variables while processing an analysis rule:

- **$value** is an iTest interpreter variable that stores the data that is extracted by the extractor. **$value** is created in the heap.
  - For the **contains** extractor (string comparisons), **$value** is either **1** (True, the string matches) or **0** (zero, False)
  - For the **regex** extractor, **$value** is the extracted value
  - For the **queries** extractor, **$value** is the result of the query
- **$itest_value** is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_value** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_value** can be overwritten by another thread.
- **$index** is an iTest interpreter variable. When the extractor extracts multiple items and the processor is invoked for each item, then **$index** holds the index of each value. For example, you would use a value's index to chart each extracted value on a separate line or series.
- **$itest_index** is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_index** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_index** can be overwritten by another thread.

Temporary data tags

- **{value}** is a temporary data tag that stores the data that is extracted by the extractor
- **{values}** is a temporary data tag that stores all of the extracted values in a space-separated list. If a value in the list includes spaces, then it is wrapped in double quotes (“). The list is not a pure Tcl list because any quotes within a value are not escaped.
- **{assertion}** is a temporary data tag that stores the assertion that is being tested by the rule. The value of assertion appears in the **Details** cell for the rule.

Contains extractor

The **contains** extractor returns the value **1** (True) if the specified string appears in the response and returns **0** (False) if the specified string does not appear in the response.

For Global rules, the **Extract using** cell displays **contains**.

For Global rules, the **What to extract** cell displays the text that is being searched on.

Contains	Specify the alphanumeric text that you wish to find or not to find in the response text.
Match type	Specify how to interpret the text in the **Contains** property. **Case-Insensitive**: The case of the text in the response is not important. **Case-Sensitive**: The case of the text in the response is important. You must specify the exact text for the Contains property and then specify Wildcard. **Regular expression**: Interpret the text in the Contains property as a regular expression. **Wildcard**: This setting indicates that the text in the Contains property includes the * wildcard character.

ExecutionDuration extractor

An analysis rule that uses the Extractor setting of **executionDuration** extracts the execution time of the current step (in seconds) and makes that available for processing.

There are no properties associated with the **executionDuration** extractor.

For Global rules, the **Extract using** cell displays **executionDuration** and the **What to Extract** cell is blank.

None extractor

An analysis rule that uses the Extractor setting of **none** is successful regardless of the response. Use this kind of rule to take an action in any case.

This is useful, for example, to produce a message or perform an assertion that is not based on extracted data. For example, your assertion might be $i > $j. This has nothing to do with extracted data, so the extractor is **none**.

For Global rules, the **Extract using** cell displays **none** and the **What to Extract** cell is blank.

Query extractor (for Global rules, Extract Using property = query)

The **query** extractor extracts data from a response by applying a mapper query (a query that is auto-generated by a response map) to the structured data associated with the response. The properties for the query extractor consist of a string containing any valid XPath or mapper

query. If the query matches multiple nodes, then each of the values of those nodes will be extracted.

For Global rules, the **Extract using** cell displays **query**.

For Global rules, the **What to Extract** cell displays the text of the query.

Query	Type the text of the query. This can be an auto-generated query or the XPath query format string. For example, the **rowCount** query represents a friendly name for the query format string **fn:count(//table/row)**
For the Query, first perform command, variable, and backslash substitutions	Check the box if the string specified for the **Query data** property uses a command field replacement, a variable, or a backslash that is used to escape a special character. As a result, the substitutions will be performed before the query is applied to the response.
Declare issue if no matches found	Check the box to specify that if the query fails to return a match, then declare an **Execution Issue** and display an execution message in the Execution view and in the test report. See the **When True** / **When False** properties.

Note For analysis rules and global analysis rules: Do not use right-click to add a query value. In Analysis rules, right-clicking and selecting **Query** to enter a query results in a query command (for example **[query . up]**), and not the intended query (for example **up**) that will return the values of interest.

- For Analysis rules, the property is the **Query** property for the query.
- For Global Analysis rules, the cell is the **What to Extract** cell.

Instead, paste or type a query from the Queries view into the **Query** property or the **What to Extract** cell.

Auto-generated queries

iTest auto-generates queries for JSON and TL1-format responses.

Mapping JSON responses See page 600.

Mapping TL1 responses See page 1216.

Regex extractor

The **regex** extractor finds all matches to the specified regular expression in the response body for the step.

For Global rules, the **Extract using** cell displays **regex**.

For Global rules, the **What to Extract** cell displays the text of the regular expression.

Regular expression	Specify the regex that will extract the data.
Use line mode	Check this box if the match always occurs within a line and does not span lines. Uncheck the box to analyze the entire response as one string.
Portion of matches to extract	**numbered_group**: Select this option to extract only a group. Specify the group number in the **Extraction group number** property. For example, in the regex **ab(c|d)fg**, **c|d** is group number 1. **full_match**: Extract all text that matches.
Extraction group number	If you selected **numbered_group** for the **Portion of matches to extract** property, then specify the number of the group here.

Declare issue if no matches found	Check the box to specify that if the query fails to return a match, then declare an **Execution Issue** and display an execution message in the Execution view and in the test report. See the **When True** / **When False** properties.
For the regular expression string, first perform command, variable, and backslash substitutions	Check the box if the string specified for the **Regular expression** property uses a command field replacement, a variable, or a backslash that is used to escape a special character. As a result, the substitutions will be performed before the regular expression is applied to the response.

Analysis rules: Properties of the processor

This topic describes the properties for each type of processor.

- In the Test Case editor, the processor is the second line in an analysis rule.
- For Global rules, the processor is displayed in the **Perform** and **Details** cells.
- In the **Response Map** wizard, the **Custom Processor** page offers the option to specify the properties described in this topic.

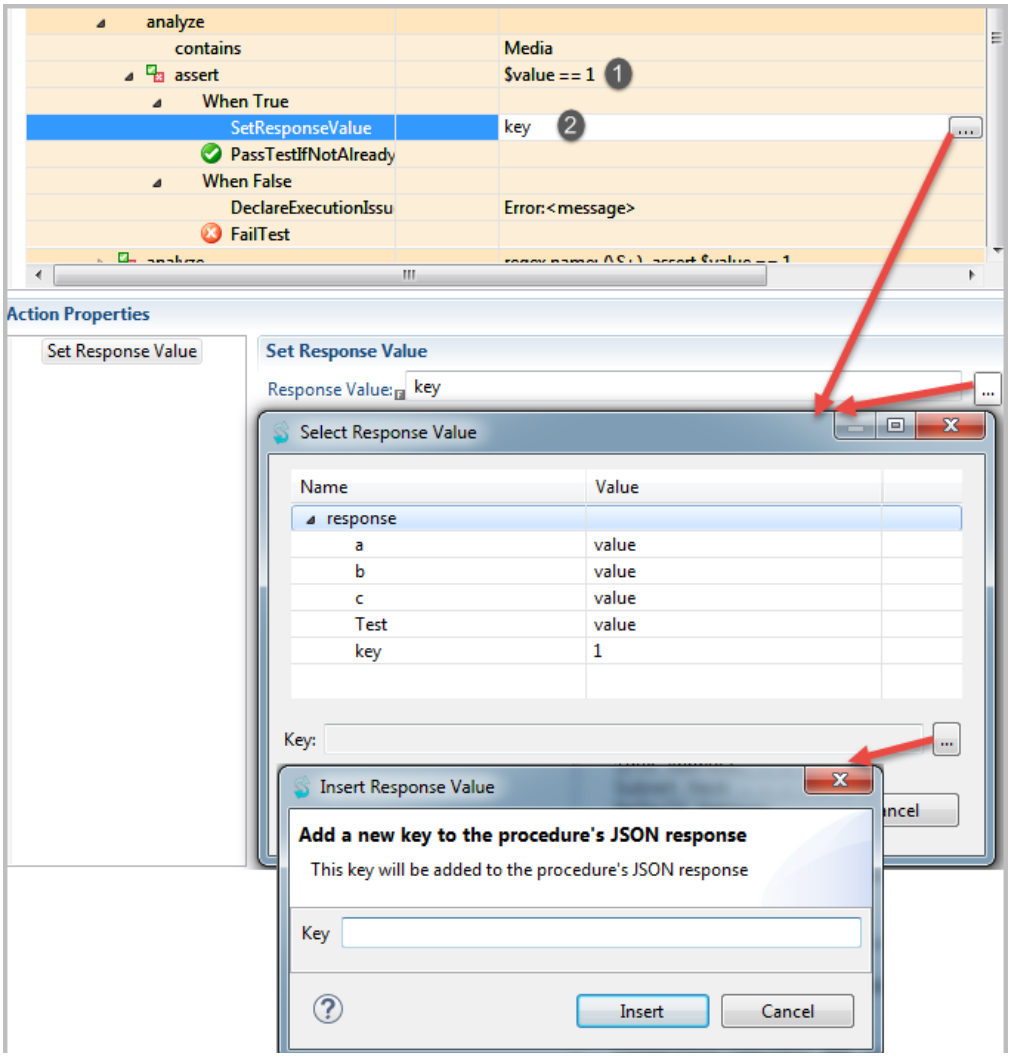
Assert processor

The **assert** processor places the extracted data into an assertion and then tests the assertion. For example, the assertion **$value == 1** tests whether the value is equal to 1.

If the value is equal to 1, then the assertion returns the value **1** (**True**). If not, then the assertion returns the value **0** (**False**). In the lines following the processor, you specify the actions that should occur based on the returned value:

- In the **When True** section, you select the **SetResponseValue** and specify the actions to take when the assertion is True (returns **1**)
- In the **When False** section, you specify the actions to take when the assertion is False (returns **0**)

Example



1 The **assert** processor places the extracted data into an assertion and then tests the assertion. The assertion appears in the **Description** cell. For example, the assertion **$value == 1** tests whether the value extracted from the response is equal to **1**.

2 If the value is equal to 1, then the assertion is True. If not, then the assertion is False.

To specify the actions that should occur upon True and False results, specify Rule Actions for **When True** and **When False**. For example, enter the Action **SetResponseValue** and specify a response in the **Description** cell. The action **SetResponseValue**, displays the **Action Properties** section.

Note You may open the **Action Properties** section in the **Properties** pane—right-click and then select the **Show Properties View** option on the menu. You may edit properties using either the **Action Properties** section (within the Test Case Editor) or via the **Properties View** tab.

You may also specify the actions response value in the **Action Properties> Set Response Value** section. In addition, you may specify the response value by clicking the ellipsis (on the Description cell or the **Set Response Value** section). iTest synchronizes the value

entered on the description cell and with the value in the **Set Response Value** section, and vice versa. See also "Select/Insert Response Value" on page 662.

For Global rules, displayed in the Details cell:

Expression
(For Global rules, displayed in the **Details** cell)
	Specify the expression to evaluate. This is typically a comparison of the query result with a particular value. Field replacements are supported.
	The expression can be any valid Tcl expression, provided that the command that you use is implemented by the iTest interpreter. You can use any of the iTest commands to create an expression using Tcl syntax. (For example, the iTest **response** command is commonly used to build an expression in assertions.)
	For the list of iTest interpreter commands, see "iTest interpreter commands in steps" on page 128.
	Predefined local variables used in assertions
	iTest populates predefined local variables while processing an analysis rule:
	• **$value** is a iTest interpreter variable that stores the data that is extracted by the extractor. **$value** is created in the heap.
	  • For the **contains** extractor (string comparisons), **$value** is either **1** (True, the string matches) or **0** (zero, False)
	  • For the **regex** extractor, **$value** is the extracted value
	  • For the **queries** extractor, **$value** is the result of the query
	$itest_value is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_value** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_value** can be overwritten by another thread.
	• **$index** is a iTest interpreter variable. When the extractor extracts multiple items and the processor is invoked for each item, then **$index** holds the index of each value. For example, you would use a value's index to chart each extracted value on a separate line or series.
	$itest_index is a Tcl interpreter variable that stores the data that is extracted by the extractor. **$itest_index** is not thread safe. Because only one instance of the Tcl interpreter is used, if you use an analysis rule in asynchronous steps, then **$itest_index** can be overwritten by another thread.
	Temporary data tags
	• **{value}** is a temporary data tag that stores the data that is extracted by the extractor
	• **{values}** is a temporary data tag that stores all of the extracted values in a space-separated list. If a value in the list includes spaces, then it is wrapped in double quotes (“). Note that the list is not a pure Tcl list because any quotes within a value are not escaped.
	• **{assertion}** is a temporary data tag that stores the assertion that is being tested by the rule. The value of assertion appears in the **Details** cell for the rule.

When multiple matches	Some extractors can return multiple values. You can set the **When multiple matches** property to perform the specified action when multiple values are returned. **Analyze each separately**: Return the result of processing each of the responses and perform the appropriate **When True** or **When False** actions for each. **Do nothing**: No errors are generated and no analysis is performed. Do not return a result of processing and continue with the next analysis rule. **True if all true**: Return (one time) the result **1** (True) only if each of the responses evaluates to True when processed. **True if any true**: Return (one time) the result **1** (True) if any of the responses evaluates to True when processed. **Fail test (raises OnAssertMultipleMatches event)**: The default actions for this option are: • Set the test result to **Fail** • Display an execution message in the Execution view and in the test report. You can configure the actions by editing the **OnAssertMultipleMatches** event (in the **Analysis Processor: Assert** group).

When True / When False

On these two processors, you specify ***two*** sets of actions to take:

- For **When True** processor, specify the actions to take when the assertion that the rule is testing returns **1** (True)

and

- For **When False** processor, specify the actions to take when the assertion returns **0** (false)

AbortExecution	Stops execution. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after an **AbortExecution** action, the **AbortExecution** action does not execute until the **FailTest** action is finished executing. No configurable properties.
AbortStep	Immediately stops executing the step and then proceed to the next step. In an analysis rule, this action has the effect of not performing remaining analysis rules and then continuing to the next step. No configurable properties.
AbortTest	Set the test result as **Abort**. Continue to execute. Note that execution is not stopped. No configurable properties.
AbortThread	Immediately stop executing the current thread. No configurable properties.
AppendToProcedureResponse	Writes the specified text to a procedure's return buffer. To return the full response, set the **Content** text to **[response .]** Supports field replacements.
AppendToStepResponse	Writes to the specified text to a step's return buffer. To return the full response, set the **Content** text to **[response .]** Supports field replacements.
Break	Break out of a **for**, **foreach**, or **while** loop. Use this **break** action to stop executing a loop and continue executing at the step after the loop. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **Break** action, the **Break** action does not execute until the **FailTest** action is finished executing. No configurable properties.

CallProcedure	Call the specified procedure (local or foreign) or the specified QuickCall. iTest provides interactive support for specifying the procedure. See “Calling a procedure in a test case step or in a property setting” on page 263. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **CallProcedure** action, the **CallProcedure** action does not execute until the **FailTest** action is finished executing. **Note** Do not add a **CallProcedure** action to either an **OnProcedureEnter** or an **OnProcedureExit** event—this results in an infinite loop. **Property:** **Procedure name and arguments**: Specify the procedure name followed by argument values if appropriate. For example, **SetupDevice -deviceNumber 3**.
Continue	The **Continue** action causes the current step to be aborted out to the innermost containing for, foreach, or while loop command. The loop then continues with the next iteration of the loop. Use the **Continue** action when you want to execute particular steps for some iterations of the loop, but not for other iterations. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event and if there are additional analysis rules listed after the current rule for the step. For example, even if a **FailTest** action appears last in the list after a **Continue** action, the **Continue** action does not execute until the **FailTest** action is finished executing. No configurable properties.

DeclareExecutionIssue	Generate an execution issue with a particular severity and display an associated execution message in the Execution view, in the Step Issues view, and in test reports. **Properties**: **Severity**: This setting specifies the type of issue: OK (executed as expected), Information, Warning, Error (did not execute as expected) **Note** In HTML, text, and XML format reports, the **OK** severity is listed as “pass” and **Error** is listed as “fail”. **Message**: Specify the text message to display in the Execution view, in the Step Issues view, and in test reports. Field replacements are supported. **Note** iTest can generate a plain language sentence for the execution message (for example, “Extracted value $value is equal to “Up”). To use this feature, specify a **Message** value of **{auto_message_true}** or **{auto_message_false}**, as appropriate. You can use any of several built-in variables to customize the message text. Information on built-in variables like **$value** appears in “Tcl interpreter local variables” on page 340. **Note** To ensure access to certain data that is available when the message is generated, iTest first applies its standard field substitution and then uses Java-style format strings for messages. Java format strings uses escaping rules that differ from Tcl rules. For example, Java string format uses single quote ' as its special character and you need two of these for escaping. So, to cause ‘ to appear in the message, use two single quotes ‘’ in the message text. For the Java string format rules, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/MessageFormat.html
Eval	Evaluate the statements specified in the **Properties** cell. No configurable properties.
ExitExecution	Immediately stop executing the current procedure and all threads to end test case execution. The **ExitExecution** action does not change the existing execution result of the test case (Pass, Fail, Abort, or Indeterminate). **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event. For example, even if a **FailTest** action appears last in the list after a **Break** action, the **Break** action does not execute until the **FailTest** action is finished executing. No configurable properties.

ExitProcedure	Stop executing the current procedure and return execution from the current procedure to the caller. You can specify a return value for the procedure in the Return value property. **Note** Because this action alters the flow of execution, it is deferred (not executed) until all other actions for the step are executed. This is true even if this action appears before other actions in the list of actions for the event and if there are additional analysis rules listed after the current rule for the step. For example, even if a **FailTest** action appears last in the list after an **ExitProcedure** action, the **ExitProcedure** action does not execute until the **FailTest** action is finished executing. No configurable properties.
FailTest	Set the test result as **Fail**. Continue to execute. No configurable properties.
PassTest	Set the test result as **Pass**. Continue to execute. No configurable properties.
PassTestIfNotAlreadyFailed	If the test result is not already **Fail**, then set the test result as **Pass**. Continue to execute. No configurable properties.
PauseExecution	Immediately pause execution. Bring the Execution view to the front. No configurable properties.
RepeatStep	Repeat the current step. The **RepeatStep** action cannot be used with an async step (a step that has the **Start this step (in a new thread) and proceed to the next step** property checked). **Properties**: **Maximum repeat count**: Specify the maximum number of times to repeat the step. **Delay between repeats**: Specify how long to wait between repetitions of the step (in seconds).
ScriptEval	Evaluate the script specified in the **Properties** cell using the Tcl interpreter attached to the execution kernel. The response is populated (including structured data) in a way consistent with how a Tcl Shell Command step works (including result, STDOUT, and/or STDERR). The following types of substitution are not made to the text of the **Command** property before the step is executed: • Command field replacements • Variables • Backslash characters used to escape special characters No configurable properties.

SetResponseValue	Configure a action response, which returns the appropriate value during test execution. For example, to specify the actions that should occur upon True and False results, specify Rule Actions for **When True** and **When False**. That is, enter the Action**SetResponseValue** and specify a response. See also "Assert processor" on page 651 (Chapter 29, "Analysis Rules: Validating Responses").
SkipRemainingRules	iTest does not perform any further analysis rules associated with the current step. For example, you might set this Action so that when an analysis rule concludes that something has gone wrong and there is no point in performing additional analysis, then skip further analysis. No configurable properties.
signal *eventName*	For synchronous execution: Wakes a thread that is waiting on *eventName* and causes it to continue execution. See "signal: Wake a thread that is waiting on an event" on page 594.
signalActivate *eventName*	For synchronous execution: Turns on the event called *eventName*. See "signalActivate: Turn a signal on" on page 595.
signalAll *eventName*	For synchronous execution: Causes the currently executing thread to sleep until all specified events have been signaled or activated (**signal**, **signalAll**, or **signalActivate**). See "signalAll: Wake all threads that are waiting on an event" on page 594.
signalClear *eventName*	For synchronous execution: Removes any instances of the event named *eventName* that had previously been activated either by a **signalActivate** step or by a **signal** command. See "signalClear: Deactivate a signal" on page 595.

- **Action Properties**

  As described in the **Actions** table, some **Actions** enable you to specify settings for properties for the action. Select a value from the list.

  Note You may open the **Action Properties** section in the **Properties** pane—right-click and select the **Show Properties View** option from the menu. You may edit properties using either the **Action Properties** section (within the Test Case Editor) or via the **Properties View** tab.

Chart processors

The **chart** processor generates charts and displays them in the Charts view. You specify the property settings for the chart in the Processors property group called **Chart Using chartType**. The property settings for the chart are summarized in the **Description** cell.

For details, see Chapter 39, "Charting Test Case Data" and See "Specifying the appearance of charts: Chart properties" on page 832..

chart_as_xy	Plot successive values of the extracted data on an XY-coordinate chart in the Charts view. Used to represent changes in numerical values over time (for example values returned in iterative loops).
chart_as_bar	Plot successive values of the extracted data on a bar chart in the Charts view. Used to represent changes in numerical values over time (for example values returned in iterative loops).
chart_as_pie	Plot successive values of the extracted data on a pie chart in the Charts view. Used to compare multiple numerical values with or without time dependency.

Execution Message processor

The **message** processor generates an execution issue and an associated message using the text specified in the **Description** cell (field replacements are supported). The message appears in the Execution view, the Step Issues view, and in test reports.

Severity **(For Global rules, displayed in the Details cell, followed by the value for the Message property)**	This setting determines the severity of the issue associated with the execution message. Select the severity of the issue: **OK** **Error** **Warning** **Information**
Message **(For Global rules, displayed in the Details cell after the value for the Severity property)**	Specify the text message to display. Field replacements are supported. **Note**: To ensure access to certain data that is available when the message is generated, iTest first applies its standard field substitution and then uses Java-style format strings for messages. Java format strings uses escaping rules that differ from Tcl rules. For example, Java string format uses single quote ' as its special character and you need two of these for escaping. So, to cause 'to appear in the message, use two single quotes ''in the message text. For the Java string format rules, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/MessageFormat.html

Add the extracted data to the test report	Check the box to add the extracted data (the data that matched the query) to the test report summary. In the Custom HTML, HTML, or XML report, the data is added to the **Extracted Data** section. The data is added in the element that you specify for the **Extracted data tag** property and lists extracted data (a single string, number, or list values corresponding to the extracted values), with one "item" for each extracted value. The following columns display: **Time:** Indicates the step execution time. **TestCase**: The filename of the test case (excluding path and suffix) that contains the step that contains the analysis rule where the message processor was invoked. **Procedure**: The procedure name containing the step. **ExecutableStep**: The executable step ID. **ExecutedStep**: The executed step ID. **Time**: The time (in floating-point seconds after execution start) when the data was exported. **Extracted Data Tag**: The extracted data item tag: boolean, number, string, etc. **Values:** Indicates the value of the extracted data. For example, true/false, a numeric value, a text string, etc.
Extracted data tag	Required if you check the **Add the extracted data to the test report** property. Specify the tag to use to identify the data that you are adding to the test report. **Note** When you select **Add the extracted data to test report** to an execution message, the tag is treated as a key, where that tag can have only one value (a single string, number, or list). Only the latest tag value (based on the execution time) You can use field replacements in the text that defines the tag. Default: [empty]

Store processor

The **store** processor stores the data that is extracted while processing the rule as a variable or a response value.

- A response with zero values or multiple values is always stored in a list.
- You can specify whether to store a single extracted value in a scalar string or in a list. See the **Always store data in a list** property for recommendations when a single extracted value can contain whitespace.
- Store processor also supports response value from the JSON response (see Procedure **Properties > Inputs and Outputs > Response** in section "Defining a procedure" on page 223 of Chapter 10, "Procedures").

Tip You can store a value from the response to a step (e.g., step 12). In a later step (e.g., step 19), you can add a rule about a token in step 19 and compare its value to the value of the token extracted in step 12. So, for step 19, you can create an assertion like: **$value > $tokenStep12 * 2**

Note You may open the context specific information—**Processor Properties** section in the **Properties** pane. Right-click and select the **Show Properties View** option from the menu. You may edit properties using either the **Processor Properties** section (within the Test Case Editor) or via the **Properties View** tab.

Variable	Specify the variable into which to store the extracted value. A response with zero values or multiple values is always stored in a list. See the **Always store data in a list** property for recommendations when a single extracted value can contain whitespace.
Global: Make the variable accessible in other procedures	Check the box to make the variable a Global variable. Global variables are available to any step in the test case. When you define a Global variable, the Data view displays the variable under the **data** node in the heap (instead of the **stack** section). This is what makes the variable. In contrast, local variables are created in the **stack** node in the heap. The **stack** section is transient, that is, it can be "popped" off and therefore lose all variable information.
Always store a single match in a list	This setting is important when you're using the response as the argument to a **foreach** statement. Specify how to store the extracted value when it is a single value. The default setting of unchecked (false) means that a single extracted value is stored in a scalar string, rather than as a list with a single element. (A response with zero values or multiple values is always stored in a list.) This setting is important when you're using the response as the argument to a **foreach** statement and a single extracted value can contain whitespace. With the default setting, a **foreach** statement that iterates over the stored variable will loop for each word in the single match, rather than once for the match. To avoid this behavior, check **Always store a single match in a list**. In contrast, if the desired behavior is to iterate over the individual words in a single match, then leave the box unchecked.
Response value	Specify the XPath that is using the extracted value to replace the sample json string (defined in **Procedure Properties > Inputs and Outputs > Response**. See section "Defining a procedure" on page 223 in Chapter 10, "Procedures"). You may enter the response value in the **Processor Properties > Store** section or click the ellipsis to select or add a new response value. See "Select/Insert Response Value" on page 662. The return value is a field substitution of [**return query_xpath**] in which the **query_xpath** is the same with **jsonSelect command** (page 380). Once the return value is defined, during execution, iTest will replace the sample json values evaluated by the **query_xpath**. Use Return Value dialog (See iTest Commands, page 384) to get the XPath easily.

writeFile processor

The **writeFile** processor writes information to a file. Here are some options for **writeFile** actions:

Note The **writeFile** action is supported for TCL only.

- You can overwrite or append to an existing file
- You can add a blank line to the end of the file (after the end of the data)
- When the information consists of multiple values, you can specify the delimiter character (comma, newline, and so on) to use to separate values
- For multiline data, you can specify the delimiter character to use to separate lines (cr, cr/lf and so on)

writeFile processor properties

If file exists	This property applies only if the specified file already exists at the time that the test case tries to write to it. Specify whether to overwrite (replace) the file or to append the specified data to the end of the file. Default: Append
Add a new line at the end of the file	Check the box to add a blank line to the end of the file after writing the specified data into the file. iTest will add the characters specified in the **Line delimiter** property Default: checked
Line delimiter	Optional. This property applies only if the specified file content consists of multiple lines. Specify the delimiter character to use to separate lines of data in the file. Default: Use the default delimiter for this platform
Value delimiter	Optional. This property applies only if the specified file content consists of multiple values. Specify the delimiter character to use to separate the values. Default: comma
Encoding	Optional. Specify the encoding type to use so that, later, it can be properly parsed. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Select/Insert Response Value

Select or insert the response value to test or store the value extracted from the test response. The dialog, **Select Response Value** or **Insert Response Value** displays when you click the ellipsis. See "Analysis rules: Properties of the processor" on page 651 ("Assert processor" and "Store processor" sections).

1 Click the ellipsis and the following dialog displays depending on whether JSON response is specified for the procedure. (**Procedure Properties> Inputs and Output > Response**). See "Defining a procedure" on page 223, in Chapter 10, "Procedures".

- The **Insert Response Value** dialog displays if no JSON response is specified.

- The **Select Response Value** dialog displays with the with the JSON response specified.

2 Select the required JSON response from the displayed list.

The value you select or will be appear on the **SetResponseValue** description cell or the **Action properties > Response Value** field.

3 Click the ellipsis on the **Select Response Value > Key** field and add a new response value, if required. You may add new response value whether a JSON response has been defined.

The text that appears in the **Description** cell reflects the value that you enter in the **Action properties > Response Value** field.

Note The response value added is saved as JSON Response in the **Procedure Properties> Inputs and Output > Response** section.

About deferred actions

Because some analysis rule actions alter the flow of execution, they are ***deferred*** — they are placed in a queue and are not executed until all other actions for the step are executed.

Such an action is deferred even if it appears before other actions in the list of actions for the event and if there are additional analysis rules listed for the step after the current rule. For example, **CallProcedure** actions are deferred because other actions may need to occur before a **CallProcedure** should happen. The table in the “When True / When False” section on page 654 notes actions that are deferred.

Deferred actions

The following actions are deferred

AbortExecution
Break
CallProcedure
Continue
ExitProcedure
RepeatStep

Example 1

The example step has two analysis rules. Each of the rules includes one action of a kind that is deferred and several other actions that are immediate (can be performed immediately).

Here is the step before execution:

step
analysis rule 1
actionA
actionB (deferred)
actionC
actionD
analysis rule 2
actionE (deferred)
actionF

When the step finishes executing, the analysis rules are evaluated in the order that they appear. When a rule is evaluated, the immediate actions are performed and all deferred actions for ***all*** analysis rules are added to the queue. After all of the immediate actions are performed, deferred actions are performed in the sequence that they were added to the queue.

Here is the order in which the actions are performed:

step
analysis rule 1
actionA
actionC
actionD
analysis rule 2
actionF
--- now that all of the other actions are finished, the deferred actions are performed ---
actionB
actionE

Example 2

Even if a **FailTest** action appears last in the list after an **AbortExecution** action, the **AbortExecution** action does not execute until the **FailTest** action and all other actions that can occur for the step are performed.

When to use a Global rule

At any time, for any step, a DUT can crash or can return a completely unexpected response. Ideally, the test case should capture the event, recognize its significance, and take appropriate actions (in the crash example, identify the crash message, collect the stack trace data, and then exit the test case or wait for a restart and then continue with a new test). You can design test cases to respond appropriately in these situations by using Global analysis rules—rules that are applied after analysis rules are applied and that act as catch-all rules.

For example, if you know that a certain class of devices generates a consistent message when it crashes, you can define a Global analysis rule for the session profile. Then, whenever iTest executes a step that references the session profile, iTest first applies any analysis rule for the step and then applies the Global analysis rule to see whether the crash condition exists. As a result, regardless of the step during which a crash occurs, it is handled by a rule that is designed specifically for the purpose.

Guidelines

- Global analysis rules work exactly like the standard analysis rules that you apply to test case steps
- There is a small delay associated with testing the assertions of Global analysis rules, so define them only if you need them
- Typically, you define the **WhenFalse** actions to ***not*** raise execution messages
- Preprocessing (to replace field replacements and so on) happens in the normal way for all rules

- Global analysis rules are applied in a clearly defined order, as described in "Precedence of Global Analysis rules" on page 665

Determining where a Global Analysis rule was triggered:

In the Execution view, the **Execution** column identifies where a Global analysis rule was applied (step, procedure, and so on). The same information appears in test reports.

Precedence of Global Analysis rules

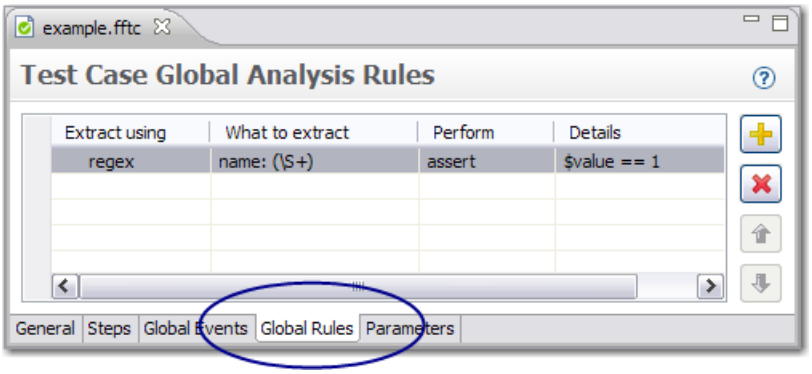
Global analysis rules are applied in a strictly controlled order. The first rules to be applied are the standard analysis rules associated with the executable step. Thereafter, Global analysis rules are applied in the following order:

- Current procedure's analysis rules
- Currently executing test case's analysis rules
- Current procedure's test case's analysis rules (for foreign procedures)
- Current session profile's analysis rules

Global Analysis Rules page

On the **Global Analysis Rules** page, you define analysis rules that apply to all steps in a test case. Analysis rules are described in "Analysis rules: Validating responses and setting Pass / Fail" on page 641.

The **Global Analysis Rules** page appears in several editors. Click the **Global Rules** tab on the editor to open the page.



How rules are applied

Based on where they are defined, rules are applied as follows.

- **Test Case editor**: Analysis rules that you define for the test case are applied to all steps in the current test case.
- **Session Profile editor** (also called the **New Session** page): Analysis rules that you define for the session profile are applied to all steps that are in a session that is based on the current session profile.

Adding and working with Global Analysis rules

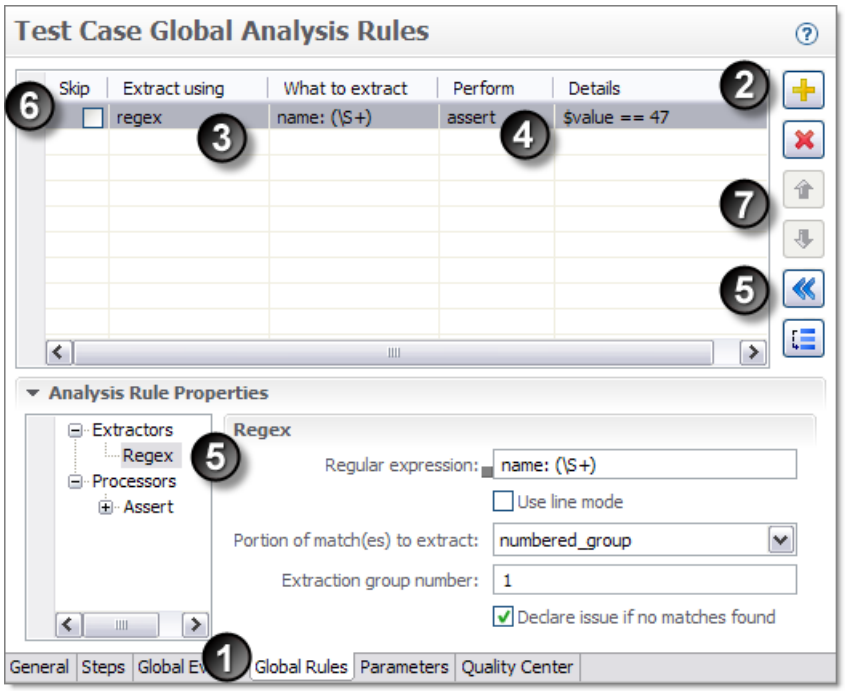
See "Adding and working with Global analysis rules" on page 666.

Adding and working with Global analysis rules

Global analysis rules are analysis rules that apply to all steps in a test case. Analysis rules are described in "Analysis rules: Validating responses and setting Pass / Fail" on page 641.

Adding a Global analysis rule

1. In the editor, click the **Global Rules** tab.
2. Click **Add** . iTest adds a default rule.
3. Click in the **Extract using** cell to select an extractor type for the rule. Based on your selection, iTest places default text in the **What to extract** cell. Replace the text with the value that the extractor should use to get the data from the response.



Important For **query** extractors, do not use right-click to insert a query value into the **What to Extract** cell. Instead, paste or type a query from the Queries view.

4. Click in the **Perform** cell to select a processor type for the rule. Based on your selection, iTest places default text in the **Details** cell. If appropriate, replace the text with the value that the processor should use to process the data that is returned by the extractor.
5. Now, you have the option to modify the property settings of the rule's extractors and processors. Click **More** to open the **Analysis Rule Properties** section. In the example, we selected **Regex** so that we could edit the properties associated with the **Regex** extractor. For details, see "Analysis rules: Properties of the extractor" on page 648 and "Analysis rules: Properties of the processor" on page 651.
6. Optional: You can use the **Skip** check box or **Skip** to skip the rule while developing/debugging a test case.

7 Optional: You can use **Move Up/Move Down** to move a selected rule in the list. Rules are applied in the listed order.

Deleting Analysis Rules

To delete an analysis rule, select it and click .

Editing Global Analysis Rules

1 To make changes to a Global analysis rule, first select it in the list.

2 Select the field that you want to edit.

3 Click the field to edit and either edit the text directly (for **What to extract** and **Details**) or make a new selection from the drop-down menu (for **Extract using** and **Perform**).

4 To modify the property settings of the rule's extractors and processors. Click **More** to open the **Analysis Rule Properties** section.

Global Analysis Rule toolbar

Add	Add a new Global rule.
Remove	Delete the selected rule.
Move Up **Move Down**	Move the selected rule one up or down in the list. Rules are applied in the listed order.
More **Less**	Open/Close the **Analysis Rule Properties** section to view or edit property settings.
Skip	You can use the **Skip** check box or **Skip** to skip the rule while developing/debugging a test case

About the Analysis Rule wizard

Analysis rules perform Pass / Fail validation on the results of a step. Use the Analysis Rule wizard to create an analysis rule that performs one of the following operations:

- Validates something about the response to a test case step (and then performs specified actions like passing or failing the test, displaying an execution message, and then charting the extracted value over time).
- Generates an execution message
- Stores data in a variable or as a JSON response value

Analysis rules generated by the wizard operate exactly like rules that you define "by hand" in the Test Case editor or that you add using the **Quick Analysis Rule** option in the right-click menu in the Response view. The wizard's last page (Summary) let's you verify the rule that you are about to create by displaying the property settings for the rule. The property names are exactly the names that you see while editing an analysis rule in the Test case editor.

You can add multiple analysis rules to a step by running the Analysis Rule wizard as often as you like.

Starting the wizard

You can start the wizard from any of the following places:

- While editing a step in the Test Case editor, right-click and select **Analysis Rule Wizard**.
- While editing a step in the Test Case editor, click **Analysis Rule Wizard** .
- In the Response view, the Structure view, or the Queries view, select the text that will get the rule (or click anywhere in the window). Then right-click and select **Analysis Rule Wizard**.

Pages in the Analysis Rule wizard

The wizard includes the following pages:

Analysis Rule Wizard: Rule page See page 668.

Analysis Rule Wizard: Select Step page See page 668.

Analysis Rule Wizard: Validation page See page 669.

Analysis Rule Wizard: Extract page See page 675.

Analysis Rule Wizard: Comparison page See page 678.

Analysis Rule Wizard: Save Data as variable or Response Value page See page 678.

Analysis Rule Wizard: Summary page See page 673.

Analysis Rule Wizard: Rule page

Specify the type of analysis rule to apply to the response.

Validate something in the response	Use this option to determine whether particular text or values appear (or do not appear) in the response text. You will next specify one of the following: • The response contains a specified string • The response does not contain a specified string • Compare the extracted value to a specified value
Create a message to display during execution	Use this option to specify an execution issue and associated **execution message** that should appear in the Execution view, Step Issues view, and Test Report editor as a result of this analysis rule.
Store data in a variable or a JSON response value	Use this option to store the returned value in a variable or a JSON response value that you will specify.
Custom	Configure an analysis rule one step at a time with the wizard's assistance.

Analysis Rule Wizard: Select Step page

The wizard displays the steps in the test case. Select the step to apply the rule to and click **Next**.

Note This page appears only if you start the **Analysis Rule** wizard from the **Develop a Test Case** activity page.

Analysis Rule Wizard: Validation page

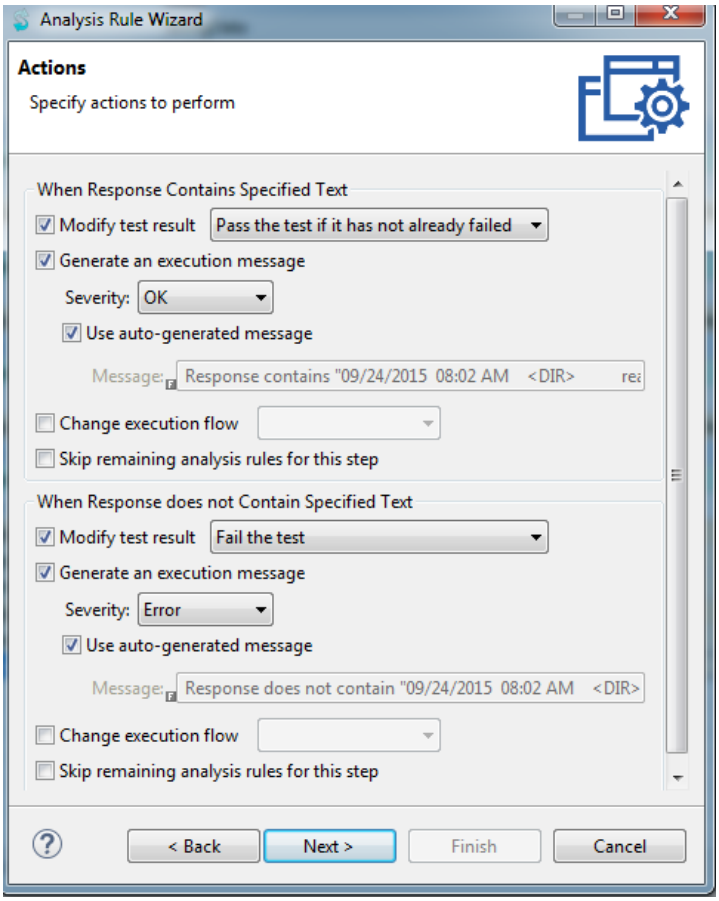
Specify how to validate the response data.

The response contains a specified string (text)	Specify the alphanumeric text that should appear in the response text. In the resulting analysis rule, the **contains** extractor returns the value **1** (True) if the specified string appears in the response.
The response does not contain a specified string (text)	Specify the alphanumeric text that should **not** appear in the response text. In the resulting analysis rule, the **contains** extractor returns the value **1** (True) if the specified string does **not** appear in the response.
Compare the extracted value to a specified value	Once iTest obtains the data from the response, compare the extracted value to an expected value that you will specify.
The response is empty	Indicates that the response value is empty.

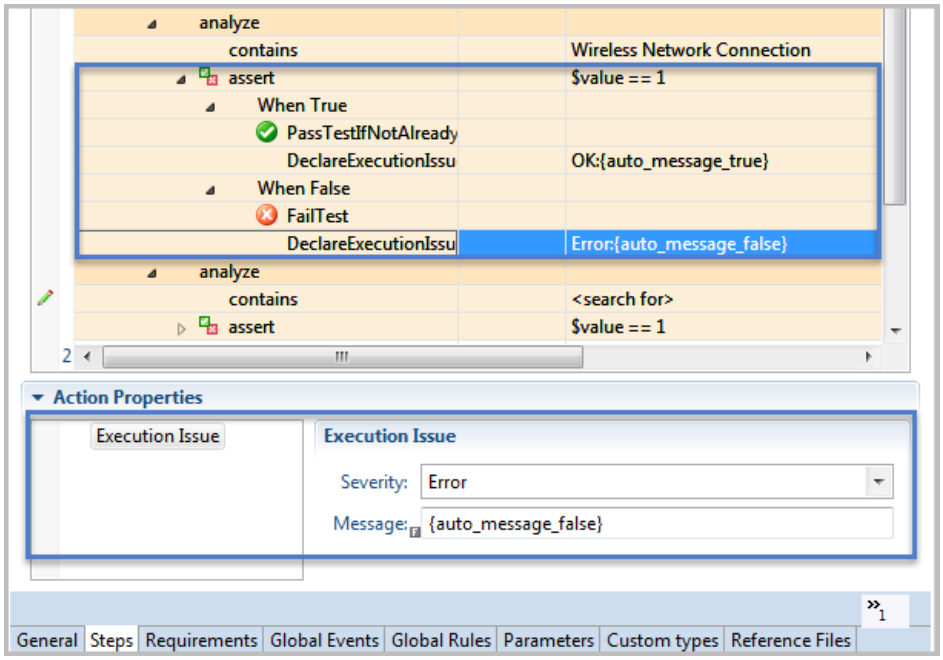
Analysis Rule Wizard: Actions page

On this page, you specify ***two*** sets of actions to take:

- The top section configures the action to take when the condition ***is met*** (for example, **When Response Contains Specified Text** as shown here)
- The bottom section configures the action to take when the condition ***is not met*** (for example, **When Response does not Contain Specified Text** as shown here)



The settings on this wizard page will generate the settings on the **WhenTrue** and **WhenFalse** properties pages in the Test Case Editor **Action Properties** property group.



The page appears in one of the following formats, depending on the method you selected for validating the response data:

When the response *should* contain a particular string, then:	• The top section is called **When Response Contains Specified Text**: You specify the actions to take when the assertion that the rule is testing returns **1** (True) **and** • The bottom section is called **When Response Does Not Contain Specified Text**: You specify the actions to take when the assertion returns **0** (false)

When the response *should not* contain a particular string, then:	• The top section is called **When Response Does Not Contain Specified Text**: You specify the actions to take when the assertion that the rule is testing returns **1** (True) **and** • The bottom section is called **When Response Contains Specified Text**: You specify the actions to take when the assertion returns **0** (false)
When you specified a comparison between the value from the response and an expected value, then:	• The top section is called **When Expression is True**: You specify the actions to take when the assertion that the rule is testing returns **1** (True) **and** • The bottom section is called **When Expression is False**: You specify the actions to take when the assertion returns **0** (false) For example, the assertion **$value == 04:00:00** tests whether the extracted value is equal to **"04:00:00"**. If the value is indeed equal to **"04:00:00"**, then the assertion is **True**. If the value is not equal to **"04:00:00"**, then the assertion is **False**. To specify the actions that should occur upon True and False results, you specify ***two*** sets of actions to take: • In the **When Expression is True** section, you specify the actions to take when the assertion that the rule is testing returns **1** (True) **and** • In the **When Expression is False** section, you specify the actions to take when the assertion returns **0** (false) Note that some extractor types can return multiple values.

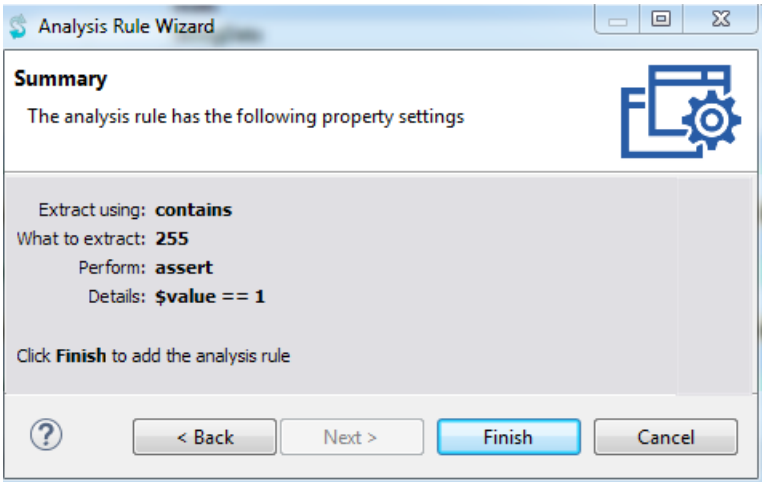
Actions

Modify test result	**Note**: Upon test case completion, the last Pass/Fail flag result applies. **Pass the test if it has not already failed**: (default setting for **WhenTrue**) If no preceding step set the Fail flag for the test case, then set a Pass flag. **Pass the test**: Even if a preceding step set the Fail flag for the test case, then set a Pass flag. **Fail the test**: (default setting for **WhenFalse**) Even if a preceding step set the Pass flag for the test case, set a Fail flag.
Generate an execution message	Check the box to display an execution message that should appear in the Execution view and Test Report editor as a result of this analysis rule. **Severity** specifies the severity of the execution issue to associate with the execution message that is displayed in the Execution view, the Step Issues view, and in test reports: **OK** **Error** **Warning** **Information**  **Unchecked**: The text in the **Message** box becomes read-write. You can change the message as needed. **Default**: Checked About the predefined variables iTest populates predefined variables while processing an analysis rule: **$value** is a iTest interpreter variable that stores the data that is extracted by the extractor. **$value** is created in the heap. For string comparisons, **$value** is **1** (True, the string matches) or **0** (zero, False) For regex, **$value** is the extracted value For queries, **$value** is the result of the query **$values** is a iTest interpreter variable that stores all of the extracted values in a space-separated list. If a value in the list includes spaces, then it is wrapped in double quotes (“). Note that the list is not a pure Tcl list because any quotes within a value are not escaped. **$index:** When the extractor extracts multiple items and the processor is invoked for each item, then **$index** holds the index of each value. For example, you would use a value's index to chart each extracted value on a separate line or series. **$itest_value** is a Tcl interpreter variable that stores the data that is extracted by the extractor.

Change execution flow	Because any action that you specify here alters the flow of execution, the action is deferred (not executed) until all other actions for the step are executed. See "About deferred actions" on page 663. **Break**: Perform the EXEC **break** action that breaks from a **for**, **foreach**, or **while** loop. The **break** action stops executing the loop and exits to the step after the loop. (Generates a **Break** action.) **Continue**: (default setting for both **WhenTrue** and **WhenFalse**) Continue execution at the next step in execution order. (Generates a **Continue** action.) **Abort the test**: Abort the test case immediately. Analysis occurs after step execution. (Generates an **AbortExecution** action.) **Exit the procedure**: Finish executing the current step and then exit the test case. If the current step is in a called procedure, then return to the calling procedure. (Generates an **ExitProcedure** action.) **Repeat the step**: Continue execution at the current step. This setting is useful for remaining at a particular point in the test until a condition is met. For example, after a DUT restart, the step could repeat until the text "System Ready" appears in the response. See **Max Repeat Count**. (Generates a **RepeatStep** action.) **Call a procedure:** Call the specified procedure (local or foreign). iTest provides interactive support for specifying the procedure. See "Calling a procedure in a test case step or in a property setting" on page 229. (Generates a **CallProcedure** action.) Specify the procedure name followed by argument values if appropriate. For example, **SetupDevice deviceNumber 3**.
Skip remaining analysis rules for this step	Check the box to cause iTest not to perform any further Analysis rules associated with the current step. For example, you might set this property so that when an analysis rule concludes that something has gone wrong and there is no point in performing additional analysis, then skip further analysis.

Analysis Rule Wizard: Summary page

The information presented here is exactly the same as the information that you see on the **Analysis Rules** properties page in the Test Case editor. This is your chance to preview the rule before you add it.



For string comparisons: **eq** means “equals” (the identical text). **ne** means “not equal to” (not the identical text).

Click **Back** to make return to earlier wizard pages to make changes.

Click **Finish** to add the analysis rule.

Analysis Rule Wizard: Custom Extractor page

On this page, you specify how to extract the data, that is, the type of **extractor** (**Extract using** property in the analysis rule).

Specify how to extract the data in the **Extract using** box. The options that appear in the **Extractor Properties** section depend upon your selection.

Contains (1 if text is found, 0 otherwise)

The **contains** extractor returns the value **1** (True) if the specified string appears in the response.

The **contains** extractor returns the value **0** (False) if the specified string does not appear in the response.

Contains	Specify the alphanumeric text that you wish to find or not to find in the response text.
Match type	Specify how to interpret the text in the Contains property. Case-Insensitive: The case of the text in the response is not important. Case-Sensitive: The case of the text in the response is important. You must specify the exact text for the Contains property and then specify Wildcard. Regular expression: Interpret the text in the Contains property as a regular expression. Wildcard: This setting indicates that the text in the Contains property includes the * Wildcard character.

Query against the structured response data

The query extractor extracts data from a response by applying a mapper query (a query that is auto-generated by a response map) to the structured data associated with the response. The properties for the query extractor consist of a string containing any valid XPath or mapper query. If the query matches multiple nodes, then each of the values of those nodes will be extracted.

Query	Type the text of the query. This can be an auto-generated query or the XPath query format string. For example, the **rowCount** query represents a friendly name for the query format string **fn:count(//table/row)**
For the Query, first perform command, variable, and backslash substitutions	Check the box if the string specified for the **Query data** property uses a command field replacement, a variable, or a backslash that is used to escape a special character. As a result, the substitutions will be performed before the query is applied to the response.
Declare issue if no matches found	Check the box to specify that if the query fails to return a match, then declare an **Execution Issue**. You will configure the message on a subsequent wizard page.

Regular expression

The **Regular expression** extractor finds all matches to the specified regular expression in the response body for the step.

Regular expression	Specify the regular expression that will match the data.
For the regular expression string, first perform command, variable, and backslash substitutions	Check the box if the string specified for the **Regular expression** property uses a command field replacement, a variable, or a backslash that is used to escape a special character. As a result, the substitutions will be performed before the regular expression is applied to the response.
Use line mode	Check this box if the match always occurs within a line and does not span lines. Uncheck the box to analyze the entire response as one string.
Portion of matches to extract	**numbered_group**: Select this option to extract only a group. Specify the group number in the **Extraction group number** property. For example, in the regex **ab(c|d)fg**, **c|d** is group number zero. **full_match**: Extract all text that matches.
Extraction group number	If you selected **numbered_group** for the **Portion of matches to extract** property, then specify the number of the group here.
Declare issue if no matches found	Check the box to specify that if the query fails to return a match, then declare an **Execution Issue**. You will configure the message on a subsequent wizard page.

Analysis Rule Wizard: Extract page

Specify how to extract the data by selecting an **Extractor** (one of the following options).

Regular expression applied to the text of the response

This selection has the effect of setting **What to extract** (the **Extractor** property) to **regex** for the resulting analysis rule.

1. When you select this option, the sample response appears.
2. Select the text in the response. The wizard applies the appropriate regular expression and displays it in the **Regex** text box.
3. Optional. Modify the regular expression if needed and then click **Next**

Query on the response

This selection has the effect of setting **What to extract** (the **Extractor** property) to **query** for the resulting analysis rule.

1. When you select this option, the wizard lists the mapper XPath queries (XPath queries that are auto-generated by the response map) and their results as they normally appear in the Queries view.
2. Select the query. The wizard displays it in the **Regex** text box.
3. Optional. Customize the **Regex** if needed and then click **Next**.

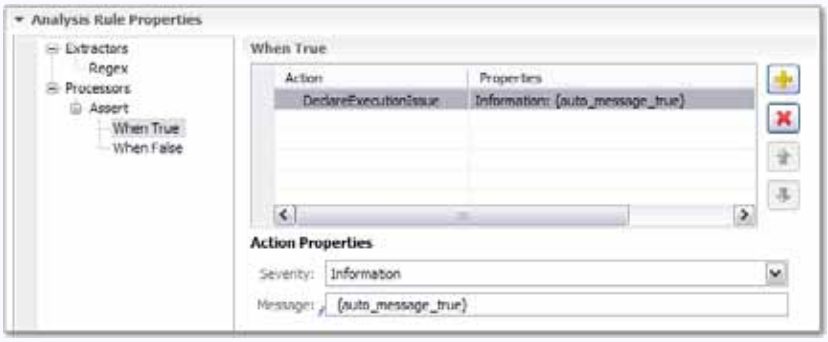
XPath query applied to the response

This selection has the effect of setting **What to extract** (the **Extractor** property) to **query** for the resulting analysis rule.

1. When you select this option, the wizard displays the structured data format of the response as it normally appears in the Structure view.
2. Select the item of interest.

Analysis Rule Wizard: Execution Message page

Because the rule will create an execution issue and associated message for display in the Execution view, the Step Issues view, and in test reports, you will now specify the details of the message. This wizard page has the effect of setting **Perform** (the **Processor** property) to **message** for the resulting analysis rule.

Severity	Specifies the severity of the execution issue to associate with the execution message that is displayed in the Execution view, in the Step Issues view, and in test reports: **OK** **Error** **Warning** **Information**
Use auto-generated message	**Checked**: The text in the **Message** box becomes dim and read-only. The box displays the message that will be generated as a plain language sentence (for example, Extracted value $value is equal to “Up”). In the resulting analysis rule as viewed in the Test Case editor, the text appears as **{auto_message_true}** and **{auto_message_false}**, as shown in this example:  **Unchecked**: The text in the **Message** box becomes read-write. You can change the message as needed. The message will appear in the **When True** and **When False** property pages for Analysis rules in the Test Case editor. The **{value}** text is replaced with the value of the built-in variable named **value**. If the extracted value is a member of a list, then you can use **{index}.** The **{index}** text is replaced with the value of the built-in variable named **index**, which holds the index of the extracted value. Use **{assertion}** to display the assertion that was tested to determine the Pass/Fail outcome. **Default**: Checked

Message	If you do not check **Use auto-generated message**, then you can specify the text message to display in the Execution view, in the Step Issues view, and in test reports. Field replacements are supported.
	The **{value}** text is replaced with the value of the built-in variable named **value**.
	If the extracted value is a member of a list, then you can use **{index}.** The **{index}** text is replaced with the value of the built-in variable named **index**, which holds the index of the extracted value.
	Use **{assertion}** to display the assertion that was tested to determine the Pass/Fail outcome.
	About the predefined variables
	iTest populates predefined variables while processing an analysis rule:
	• **$value** is a iTest interpreter variable that stores the data that is extracted by the extractor. **$value** is created in the heap.
	  • For string comparisons, **$value** is **1** (True, the string matches) or **0** (zero, False)
	  • For regex, **$value** is the extracted value
	  • For queries, **$value** is the result of the query
	• **$values** is a iTest interpreter variable that stores all of the extracted values in a space-separated list. If a value in the list includes spaces, then it is wrapped in double quotes (“). Note that the list is not a pure Tcl list because any quotes within a value are not escaped.
	• **$index:** When the extractor extracts multiple items and the processor is invoked for each item, then **$index** holds the index of each value. For example, you would use a value's index to chart each extracted value on a separate line or series.
	• **$itest_value** is a Tcl interpreter variable that stores the data that is extracted by the extractor.

Add the extracted data to the test report	Select **Add the extracted data to the test report** and enter a value (a single string, number, or list) for the **Extracted Data Tag** (mandatory). When an execution message adds extracted data to the test report, the tag is treated as a key, where that tag can have only one value (a single string, number, or list). When a test case generates multiple execution messages with the same **Extracted Data Tag**, only the latest tag value will show in the test report.

Limitations

The following limitations apply for the data extracted for each execution:

- **Total elements stored:** A maximum of 128 extracted data items per execution.
- **Bytes stored:** A maximum of 128 characters of any tag or value. Any tag or value that exceeds 128 characters will be truncated.
- **Array elements stored:** Any extracted data item whose value is an array that exceeds 128 items will be rejected (discarded).

Analysis Rule Wizard: Save Data as variable or Response Value page

Specifying whether to save the extracted data as variable or response value

Because the rule will store the extracted data as a variable or a response value, you will now configure the variable. This wizard page has the effect of setting **Perform** (the **Processor** property) to **store** for the resulting analysis rule.

Variable name	Specify the variable into which to store the extracted value. A response with zero values or multiple values is always stored in a list. See the **Always store data in a list** property for recommendations when a single extracted value can contain whitespace.
Global: Make the variable accessible in other procedures	Check the box to make the variable a Global variable. Global variables are available to any step in the test case. When you define a Global variable, the Data view displays the variable under the **data** node in the heap (instead of the **stack** section). This is what makes the variable. In contrast, local variables are created in the **stack** node in the heap. The **stack** section is transient, that is, it can be "popped" off and therefore lose all variable information.
Always store single match in a list	This setting is important when you're using the response as the argument to a **foreach** statement. Specify how to store the extracted value when it is a single value. The default setting of unchecked (false) means that a single extracted value is stored in a scalar string, rather than as a list with a single element. (A response with zero values or multiple values is always stored in a list.) This setting is important when you're using the response as the argument to a **foreach** statement and a single extracted value can contain whitespace. With the default setting, a **foreach** statement that iterates over the stored variable will loop for each word in the single match, rather than once for the match. To avoid this behavior, check **Always store a single match in a list**. In contrast, if the desired behavior is to iterate over the individual words in a single match, then leave the box unchecked.
Response value	Specify the XPath that is using the extracted value to replace the sample JSON string defined in the **Procedure properties > Input and Outputs > Response** ("Defining a procedure" on page 223, in Chapter 10, "Procedures"). The return value is a field substitution of [**return query_xpath**] in which the **query_xpath** is the same with **jsonSelect command** (iTest Commands, page 380). Once the return value is defined, during execution, iTest will replace the sample json values that is evaluated by the **query_xpath**, see "Store processor" on page 660 for more details

Analysis Rule Wizard: Comparison page

Specifying what to compare the expected value to.

Step 1. Specify the type of comparison

Specify how to compare the data from the response to an expected value. You can specify either a string or a numeric comparison.

Comparison type	Specify the method for comparing the two values. equal to does not equal greater than greater than or equal to less than less than or equal to within a specified range (inclusive) within a specified range (exclusive) outside a specified range (inclusive) outside a specified range (exclusive) matches a Wildcard string (using *, ?) does not match a Wildcard string (using *, ?) matches a regular expression does not match a regular expression matches one value in a list of values does not match any values in a list of values
Use string-based rather than numeric comparisons	Check the box to compare two string (text) values. Uncheck if the values are numeric.

Step 2. Specify the expected value or values

Specify the value that you want to compare the value from the response to.

Using	This value tells the wizard how to interpret the value that you specify in the **Value** field. **A specified value**: The value in the **Value** field is a particular string value or numerical value **The name of a local variable**: The value in the **Value** field is a local variable to have a particular string value or numerical value. **The value of a Global variable**: The value in the **Value** field is a Global variable to have a particular string value or numerical value. **The name of an argument to this procedure**: The value in the **Value** field is the name of an argument to the procedure. (The selected step is in the procedure.) **The value of a parameter**: The value in the **Value** field is the name of the parameter to evaluate. **The name of a parameter**: The value in the **Value** field is the name of a parameter.
Value	Specify the value to compare the response data to. If you selected data in the sample response before starting the wizard, then it appears here.

Step 3. (Optional) Customize the expression to evaluate

This section displays the expression that the rule will test.

The wizard auto-generates the expression based on your selections in Steps 1 and 2. Typically, you do not need to modify the expression.

Analysis Rule Wizard: Text page

Because the rule will determine that certain text must (or must not) appear in the response, you will now specify the text.

1. In the **Response text** select the text of interest.

   Alternatively, you can type the text if it does not appear in the current sample response. Type the text into the **Text that must [must not] appear** field.

   The wizard adds the text to the **Text that must [must not] appear** field.

2. Optional. Modify the text if needed and iTest will use the resulting text to perform the comparison.

Analysis Rule Wizard: Processor page

On this page, you specify how to process the data, that is, the type of **Processor** (the **Perform** property in the analysis rule).

The processor performs an action with the data. For example, you use the **chart** processor to generate charts and the **assert** processor to make comparisons between an expected value and the value returned during testing.

Perform

The options that appear in the **Processor Properties** section depend upon your selection in the **Perform** field.

For details on the properties available for each of the selections, see the appropriate section in "Analysis rules: Properties of the processor" on page 651.

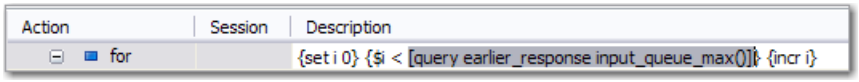
Compare and take action accordingly	Sets the **Perform** property to **assert** for the resulting analysis rule
Chart data on X-Y plot	Sets the **Perform** property to **chart_as_xy** for the resulting analysis rule
Chart data using bar chart	Sets the **Perform** property to **chart_as_bar** for the resulting analysis rule
Chart data using pie chart	Sets the **Perform** property to **chart_as_pie** for the resulting analysis rule
Store the extracted data in a variable	Sets the **Perform** property to **store** for the resulting analysis rule
Display an execution message	Sets the **Perform** property to **message** for the resulting analysis rule

Applying queries to stored responses

Often, you will want to use a value taken from the response to a step that executed earlier in the test.

In this example, the response to an earlier step included the setting for the maximum allowed input queue for the device. Now, you want to use the **input queue** value in the current step as the controlling limit in a **for** loop. To do this, you'll insert a query into the **for** statement that will return the value from the stored response to the earlier step.

In this **for** statement, we use a **query** command as a field replacement to supply the upper limit of repetitions for the loop. The value that we need appeared in the response to a step that executed earlier in the test.



The response for the earlier step had been stored in a variable named **earlier_response**.

The query that can return the value from the response is **input_queue_max()**.

If the query returns a value of **75**, then, at runtime, the **for** statement becomes:

```
{set i 0} {$i < 75} {incr i}
```

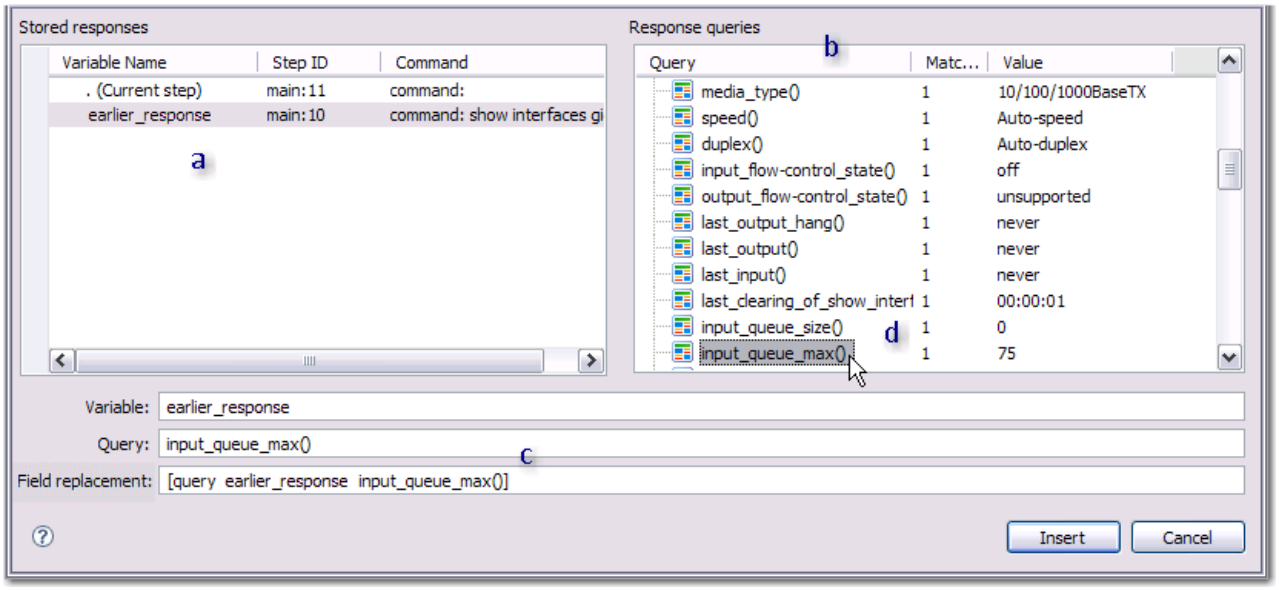
This topic provides instructions for the quickest way to insert a field replacement for a **query** command applied to a stored response. At runtime, the field replacement is replaced by a value returned from the response for an earlier step.

Inserting a query that returns a value from a stored response

1. For the step whose response includes the value of interest, ensure that the response is mapped — by a response map, an analysis rule that you added manually, or because the response is structured (for example, SNMP, JSON, or TL1).
2. For the step, store the response in a variable (as described in "Storing a response into a variable (for use later in the test)" on page 140). To ensure that the structured part of the response is stored, do not check the **Store only the text of the response** check box.
3. Now, later in the test, create the step that will use the value (for example, a **for** statement or an **eval** comparison statement).

4 Insert the query. In the **Description** cell or the **Command** field, place the cursor where the query should appear (you can select text that will be replaced), and then right-click and select **Insert > Query on Stored Response**.

The **Insert Query on Stored Response** dialog box opens.



a All responses (for the test case) that are stored in variables appear in the **Stored Responses** list. Global variables are displayed as ***/data/****varName*.

Select a stored response.

b When you select a stored response, all of the queries that are defined for the response appear in the Response Queries list. Select a query.

c When you select a query, the text in the **Field replacement** text box is updated with the appropriate **query** command in the standard format: **[query** *varName mapperQuery***]**.

d Double-click a **Query** in the list or select one and then click **Insert**. The text in the **Field replacement** text box is then added at the position of the cursor.

Note Even though the response to the current step may appear in the **Stored Responses** list, it makes no sense to select it for this purpose – the step will not have been executed when the field replacements are made and there is therefore no response for the step.

If a special character (“ **\ [ ] $** or the space character) appears in the query, then, in the **Field replacement** text box, the **** character is inserted to escape the special character. The result is a properly-formatted field replacement.

CHAPTER 30

The iTest Builder

Building projects

The ***builder*** process ensures that all resources (iTest files) that depend on each other are properly linked so that updates to one resource are reflected in all of its dependent resources. This ensures that tests will execute correctly. The builder detects dependency problems and either fixes them or displays Warnings or Errors in the Problems view.

Examples

- A test case refers to a particular response map (the test case has a dependency on the response map). When you delete the response map, the builder notices that a required document no longer exists and notifies you by creating an error message in the Problems view.
- If a test case is dependent on a reusable procedure library that changes frequently, the builder updates the test case to stay current with the changes.

By default, iTest builds projects automatically to update dependency declarations as needed. We strongly recommend that you leave automatic builds enabled (**Project > Build Automatically**)

The Dependencies view displays dependency relationships in both directions. The view displays the information for either the file in the active editor or the currently selected files in the iTest Explorer or Favorites view. If you select multiple files, then the view displays dependency information for each file. See “Dependencies view” on page 689.

Important When you delete a project, all other projects in the workspace are rebuilt to add error markers to the projects that reference resources that had previously existed in the deleted project. If you then create a new project with the same name as the deleted project, iTest does not perform another build (for performance reasons) and the error markers in the projects remain (incorrectly).

To avoid this possible problem, when you add a new project or delete a project, perform a clean build (click **Project > Clean > All**).

Refactoring iTest resources (renaming, moving, or deleting files)

Typically, any iTest resource (a iTest file) depends upon one or more other iTest resources. For example, a test case might depend on a topology, response maps in a response map library, session profiles, and so on. As a result, when you **Rename**, **Move**, or **Delete** a iTest resource, multiple files will typically be affected. For any of the following resource types, iTest auto-updates dependency relationships when you make such changes:

- Test cases
- Session profiles
- Response maps
- Parameter files
- Topologies
- Testbeds
- Job files
- Test suites

Note iTest supports the Undo operation for any of the changes mentioned.

Restrictions

- **Folders and projects are not auto-updated**

  iTest does not auto-update dependency relationships when you rename, move, or delete folders or projects.

- **Procedure name changes and device name changes are not auto-updated**

- **Drag-and-Drop moves are not auto-updated**

  To ensure that dependency relationships are auto-updated, right-click the resource and then select **Move**.

To rename, move, or delete a iTest resource

1. Select the file (or multiple files using Ctrl-click or Shift-click).
2. Right-click the selection and then select **Rename**, **Move**, or **Delete**.
3. In the **Rename Resource** or **Move Resource** or **Delete Resource** dialog box, click **Preview** to see the structured list of all iTest resources that will be affected by the change. Individual

references to the resource that will change (for example, a run step to a test case that is being renamed) are listed under each file.

By default, all affected resources are selected. You have the option to uncheck the resources that should not be updated with the new name or location of the changed resource.

Note All references to a deleted resource become validation warnings in the affected dependent file (because there is no way to update a reference to a deleted file).

4 Click **OK** to apply the changes.

Searching and replacing property values in iTest files

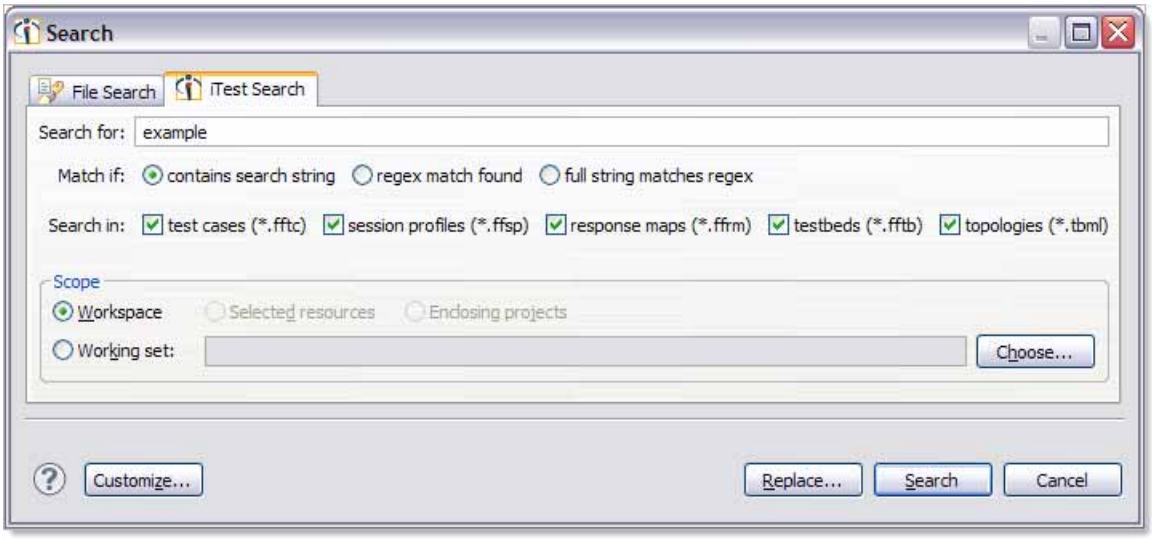
You can search for and (as needed) replace property settings across any type of iTest resource. For example, you can search/replace a hard-coded IP address setting with field replacement text that enables the IP address to be set at execution time — change the text **10.123.4.5** to **[param routerIP]**. You can update the following types of iTest resource:

- test case
- session profile
- response map
- testbed
- topology

Note All searches are not sensitive to case (case-insensitive).

To Search and replace property values

1 On the iTest menu, click **Search > Search**. On the **Search** dialog box, click the **iTest Search** tab.



Search for •To search for files of the type that you specify in the Search in section, leave the **Search for** field empty.

- To search for text in files, type the text search expression.

  If you specify **Match if** as **contains search string**, then use any of the following wildcard characters:

  ***** matches any set of characters, including the empty string

  ? matches any character

  **** is the escape for a literal. To search for an asterisk, question mark, or backslash character, type a backslash before it to indicate that you are not using the character as a wildcard ("*", "\?", or "\\")

Match if Specify the type of search.

Search in Select all of the resources (file types) to search through.

Scope Specify the scope to search: the whole workspace, pre-defined working sets, previously selected resources, or projects enclosing the specified resources.

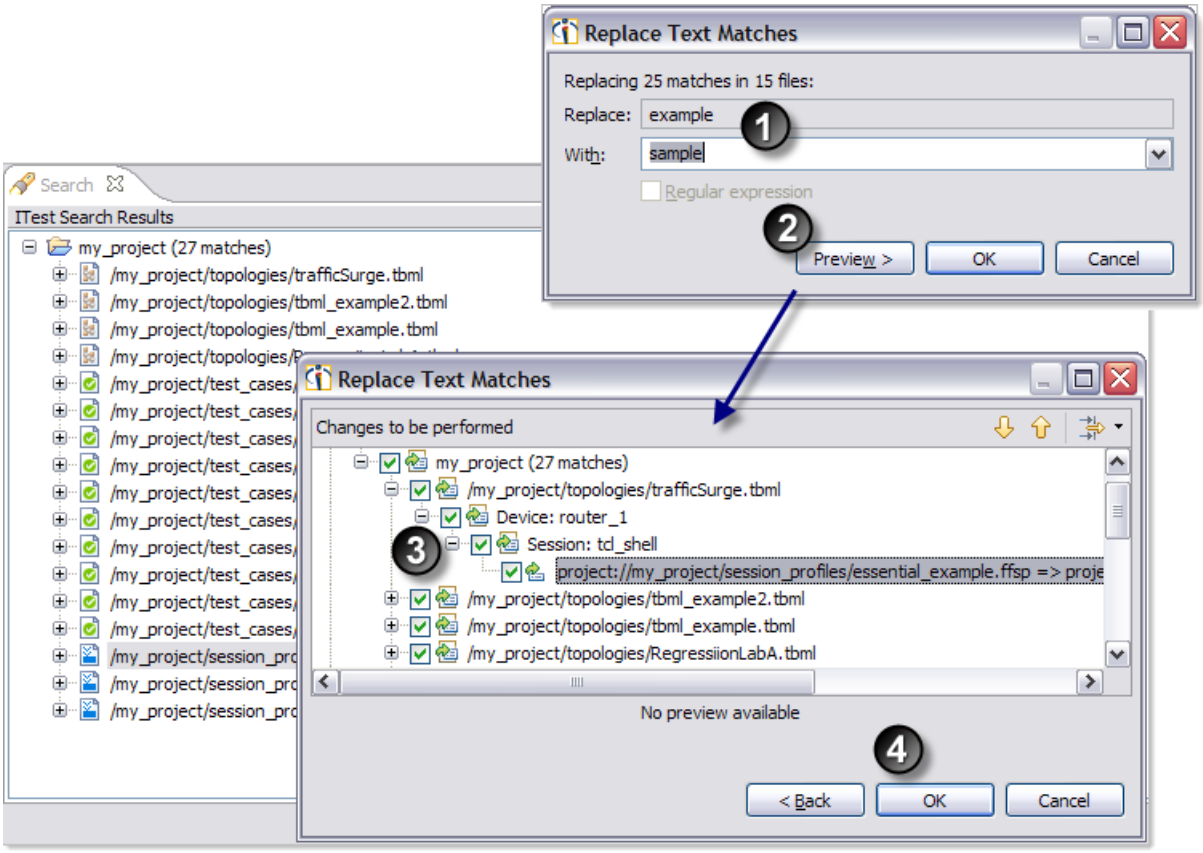
2 Click a button to perform an action:

Customize Click **Customize** to specify the types of files to search — iTest resources or all file types.

Replace To replace the specified search text, click **Replace**. In the **Replace Text Matches** dialog box, specify the text to replace the search text with ❶.

For global replacement, click **OK**. To select the instances of the text to replace, click **Preview** ❷. By default, in the **Replace Test Matches** dialog box, all instances are selected for

replacement 3. Uncheck instances that should not be replaced. Click **OK** to perform the replacements 4.



Search Click **Search** to search for all instances of the text. The results of the search appear in the Search view. Double-click an instance to view the property setting in the appropriate editor.

Specifying the projects that your project depends on

To enable iTest to generate an itar file correctly, you must specify all projects that any file in the selected project depends upon. For example, a test case in the selected project depends on a response map in a particular response map library (remember that response map libraries are projects).

1 Before you start, build the project relationships.

In the iTest Explorer, right-click the project and select **Properties**. (Alternatively, select the project and select **Project > Properties** in the main menu.)

2 Select **iTest Project Dependencies**.

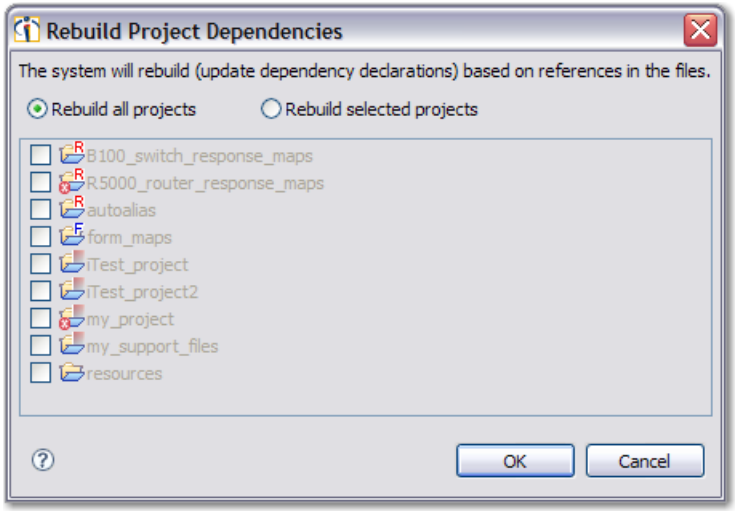
3 Select each of the projects that any file in the selected project depends upon.

4 Click **OK**.

Rebuilding project dependencies

During the builder process, iTest goes through each specified project. If a file in a project refers to another project, then the other project is declared as a dependent project. This is important when you are creating itar files, as described in “Building projects” on page 683.

1 In the **Project** menu, select **Rebuild Project Dependencies.**



2 Select the appropriate option and then click **OK**.

- **Rebuild all projects**:
- **Rebuild selected projects**: To speed the process, build only the selected projects.

Setting preferences for the builder

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > General > Builder.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > General > Builder

The builder process ensures that all files and resources that depend on each other are properly linked so that updates to one file are reflected in all of its dependent files. The builder "builds" projects to catch dependency errors before you try to execute a test that will probably fail due to the error.

Here is an example of how the builder detects a dependency problem: A test case refers to a particular response map (the test case has a dependency on the response map); when you delete the response map, the builder notices that a required document no longer exists and notifies you by creating an error message in the Problems.

By default, iTest builds projects automatically to update dependency declarations and to fix problems as needed. We strongly recommend that you leave automatic builds enabled (**Project > Build Automatically**).

Suppress warnings for circular project dependencies	If your workspace includes many circular references, iTest might create so many warnings in the Error Log that the system cannot operate. Example: Project A includes file A that refers to file B in project B. File B in project B refers to file A in Project A. The iTest validator code goes into a loop to create an infinite number of warnings. For this case, check the box so that iTest does not generate the warnings. After you check the box, existing warnings will remain until you rebuild the projects (click **Project > Clean > Clean all projects**) Default: unchecked

Dependencies view

When an **open** step in test case "A" refers to a particular session profile, we say that the test case depends on the session profile — it has a dependency relationship with the session profile. Similarly, the test case might depend on the default testbed that it references on the **General** page. Because we know about the dependencies, we can predict that if the either the session profile or testbed were moved, renamed, or deleted, then the test case could not execute (a file that the test case depends on would not be available at runtime).

In addition, dependency can go in the other direction: other files might depend on test case "A". For example, test case "B" might include a **call** step that calls a procedure that is defined in test case "A" — test case "A" is referenced by test case "B".

So, one file can both depend on other files and also be depended upon by (that is, be referenced by) yet other files.

The Dependencies view displays dependency relationships in both directions. The view displays the information for either the file in the active editor or the currently selected files in the iTest Explorer or Favorites view. If you select multiple files, then the view displays dependency information for each file.

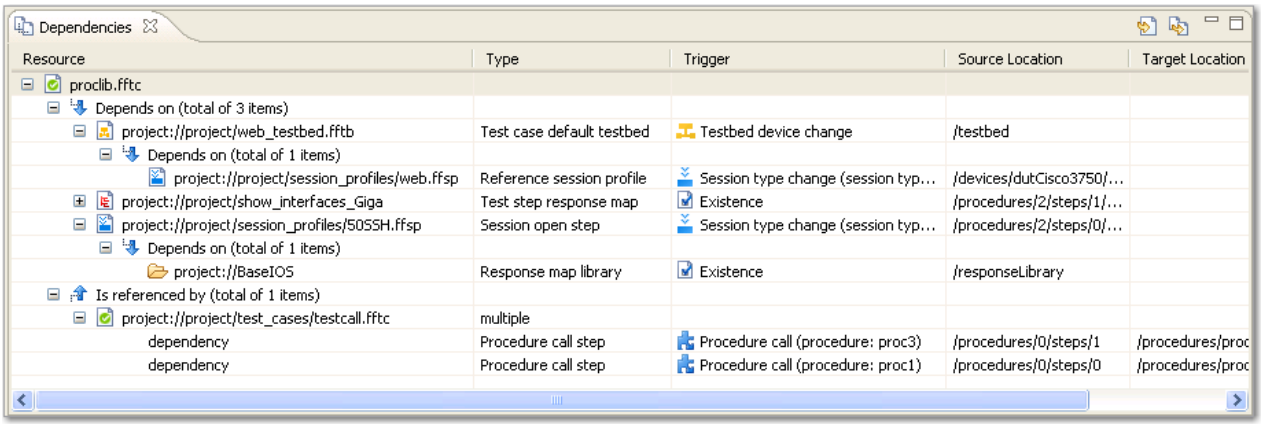
Example

In this example, we view the dependency information for a single file: the **proclib** test case.

You can see that **proclib** depends on the **web_testbed** testbed, on the **show_interfaces_Giga** response map, and on the **50SSH** session profile. In this context, we say that **proclib** is the ***dependent*** file and that **web_testbed** is a file that ***causes the dependency***.

Notice that the files that **proclib** depends on also have files that they, in turn, depend on. The **50SSH** session profile depends on the **BaseIOS** response map library. To learn which files the **show_interfaces_Giga** response map depends on, we could click ⊞ next to its entry.

In addition, the view shows that the **proclib** test case is referenced by the **testcall** test case, that is, the **testcall** test case depends on the **proclib** test case. It seems that **testcall** has multiple dependencies on **proclib**: it includes two **call** steps that call procedures that are defined in **proclib**.



Columns on the view

Resource	The Dependencies view displays dependency information for the top-level file or any resource that appears in the **Resource** column. • If you select a single file or select an editor, then the single file appears at the top level in this column. In the example, only one file — the **proclib** test case — is at the top level. • If you select multiple files in either the Test Explorer or Favorites view, then each of the files appears at the top level in this column, one file after the other. Expand any group by clicking ⊞ to see the dependency information for the resource. Double-click any file in the view to open the file in an editor. The Dependencies view immediately refreshes to display the dependency information for the just-opened file. **Depends on** section For each dependent file, the view displays a **Depends on** section that lists each resource upon which the file depends. In the example, you can see that the **proclib** test case depends on three resources: a testbed, a session profile, and a response map library. The view can display any number of levels of dependency — in the example, the testbed depends on a session profile. Click ⊞ for a file to view the resources that cause the dependencies. **Is referenced by** section The **Is referenced by** section lists all files that include a reference to the top-level file. In the example, the **testcall** test case refers to the **proclib** test case by calling procedures that are defined in **proclib** (that is, the **proclib** test case is depended upon by the **testcall** test case).
Type	In the **Depends on** section, the **Type** column displays the dependency relationship — why the dependent file depends on the file that causes the dependency. In the **Is referenced by** section, the **Type** column displays the reason that the file that references the original (dependent) file depends on the original file.

Trigger	The **Trigger** represents the kind of change that would either: • Cause the system to perform a build (build the documents) to ensure that dependencies are maintained. In the example, **web_testbed** has a **Device change** Trigger. If there were a change to a device in the **web_testbed** testbed, then the system would perform a build to ensure that **proclib** would execute properly. or • Cause the file to be unable to perform its function. In the example, **web_testbed** has an **Existence** Trigger. If the **web_testbed** response map were n longer in existence (it was deleted or renamed), then **proclib** could not execute. **Existence**: The dependent file is affected if the file is renamed, relocated, or deleted. **Change**: A change in the file would result in a system rebuild to ensure correct operation of the dependent file. **Session type change**: A change in the session type for the session profile could cause incorrect operation of the dependent test case **Device change**: A change to a device definition would require a rebuild to ensure proper test case execution. **Procedure call**: A change in the call step could affect the associated file.
Source Location	The location in the dependent file that results in the dependency (such as the EXEC **call** step that refers to a procedure in a foreign test case). When you double-click a resource (or select a resource and click **Go to Resource**), the editor opens the file to this location.
Target Location	Applies to **call** steps only. location in the dependent file (such as the procedure that is being called)

Dependencies view toolbar

Go to Resource	Open the selected file in the appropriate editor. Alternatively, double-click a resource to open it in an editor.
Go to Connected Resource	Open in the appropriate editor, the associated dependent file (the file that depends on the selected file).

Using the Dependencies view

The Dependencies view is useful in situations where you need to know how a change in one file affects another file. For example:

- You are reorganizing, renaming, or moving a group of tests as a part of a larger “cleanup” operation in your organization (this is often called *refactoring*). In some cases, you will delete duplicate tests. Use the Dependencies view to determine which files will be affected by a change. If a required file is not found in the expected location, the view displays “(does not exist)” for the missing file.
- You would like to share a test case with a colleague and will therefore give them all files that the test case requires in order to execute properly. Use the Dependencies view to determine what to share. You should share entire projects (as opposed to individual files) so as to preserve the file structure that the test expects.

CHAPTER 31

iTestCLI (Obsolete and Deprecated)

Important

iTestCLI is ***obsolete and is no longer under development***.

This chapter is provided only to support existing implementations. You should develop new test cases using iTestRT (described in Chapter 32, “iTest Runtime: iTestRT”).

Using itestcli

iTestCLI is the original implementation of the command line (or “headless”) version of iTest. iTestCLI can operate only on iTest files that are held in a iTest workspace. The more flexible implementation is iTestRT, as described in Chapter 32, “iTest Runtime: iTestRT”.

- To execute iTestCLI on Linux, you must specify the full path to the executable at the command line or source the script and type `itestcli.sh`
- To execute iTestCLI on Windows, type `itestcli.bat`

Running itestcli

To execute a iTest test case from the command line, you enter a command of the following form:

```
itestcli -w workspace_dir test_case_URI_or_path
```

Example 1

```
itestcli -w "c:/itest/ws1" "project://project1/folder1/tc1.fftc"
```

or

```
itestcli -w "c:/itest/ws1" "path_somewhere_on_filesystem/folder1/tc1.fftc"
```

- The first form of the command asks to execute the test case at the specified URI — a file inside the **project1** project inside a workspace whose root is on the file system at **c:\itest\ws1**
- The second form of the command specifies a path for the test case rather than a URI

  If you do not specify a workspace, iTest determines the workspace that the test case file resides in using the following process. For this option to work, you must have opened iTest at least one time so that the workspace can be determined.

a Move up the directory structure of the specified file to determine whether the file resides in a subdirectory within a iTest workspace.

b If the system finds no iTest workspace, then it tries each workspace in the workspace history (the list of workspaces that the user has opened using **File > Switch Workspace**).

c If the system still finds no iTest workspace, then it displays an error message.

Example 2

This example shows the arguments that an actual command might include (command prompt in blue text). Arguments are fully described in "itestcli command reference" on page 695.

```
C:\Documents and Settings\[username]\My Documents\iTest_4.4_workspace\
my_project\test_cases> itestcli -p loop_count=5 -w "C:\Documents and
Settings\[username]\My Documents\iTest_4.4_workspace" -t HTML -r "C:\temp"
-es project://my_project/test_cases/favorite_testcase.fftc
```

Notes:

- For Windows, forward slashes or backslashes are acceptable in the workspace path.
- Use `itestcli --help` (or `itestcli -h`) for online help
- Progress is shown at the command line.
- For test cases that include, for e.g., SNMP sessions, you must use the `–guiserver` option to start iTest in full UI mode.
- The Eclipse GUI may pop up if the execution involves a session that executes using the iTest browser. Otherwise, no GUI should pop up.

If there are two shells open, you can execute two test cases in the same workspace at the same time (or you could execute the same test case twice at the same time). All of this depends on the notion of a "workspace server" that does the actual work.

Error cases

The workspace may be open in the GUI or the workspace server will refuse to start. In this case, you get a message at the console: "You must close the interactive iTest session in order to use itestcli."

A specified file might not be in a workspace that iTest can locate.

Differences between iTestRT and itestcli

While both itestcli and iTestRT can be used for executing tests from the command line, there are some fundamental differences.

- Think of itestcli as a command line wrapper on top of iTest – when it boots, it loads up all the iTest modules, including all the GUI components used for test authoring which are then suppressed after being loaded into memory.
- In contrast, iTestRT is the much leaner iTest execution engine extracted from iTest, without any of iTest's test authoring and management capabilities. While iTestRT is limited in certain areas, iTestRT will eventually include all features and will completely replace itestcli.

HP ALM (formerly Quality Center) integration

iTest's ALM integration supports both iTestRT and itestcli for test execution and test report publishing to HP-ALM.

itestcli command reference

All commands except test report database commands are described in the table titled "itestcli command-line arguments" on page 697.

Test report database commands are described in "Specifying an external database for test reports" on page 702.

Usage

```
itestcli [options] [testcaseURI [testcaseURI [testcaseURI] ...]]
```

Options (arguments) are fully described in the table.

To execute a iTest test case from the command line, enter the following command:

```
itestcli -w workspaceDir testcaseURI
```

Example 1

```
itestcli -w "c:/itest/ws1" "project://project1/folder1/tc1.fftc"
```

or

```
itestcli -w "c:/itest/ws1" "path_somewhere_on_filesystem/folder1/tc1.fftc"
```

- The first form of the command asks to execute the test case at the specified URI — a file inside the **project1** project inside a workspace whose root is on the file system at **c:\itest\ws**
- The second form of the command specifies a path for the test case rather than a URI

  If you do not specify a workspace, iTest determines the workspace that the test case file resides in using the following process. (If you do not specify a workspace, then, before using itestcli, you must open iTest at least one time. This populates the workspace information so itestcli can determine the workspace.)

  a Move up the directory structure of the specified file to determine whether the file resides in a subdirectory within a iTest workspace.

  b If the system finds no iTest workspace, then it tries each workspace in the workspace history (the list of workspaces that the user has opened using **File > Switch Workspace**).

  c If the system still finds no iTest workspace, then it displays an error message.

Example 2

Execute test case with minimal options:

```
itestcli -w "c:\itest\ws1" project://my_project/test_cases/tc1.fftc
```

or

```
itestcli c:\itest\ws1\my_project\tc1.fftc
```

Example 3

Execute with runtime parameters and export HTML test report to reports folder:

```
itestcli -w "c:\itest\ws1" -p "param1=my value1" p2=value2 -r
"c:\itest\reports" -t HTML project://my_project/test_cases/tc1.fftc
```

Example 4

Execute two test cases in GUI mode with a testbed and parameter file from project named `proj1`:

```
itestcli -w "c:\itest\ws1" -g -b project://proj1/tb1.fftb -P
project://proj1/parameters.ffpt project://my_project/test_cases/tc1.fftc
project://my_project/test_cases/tc2.fftc
```

Example 5

Execute a test case then render the resulting test report into an HTML file in a particular directory (this example uses the **–t** and **–r** options).

```
C:\>itestcli -w "c:\documents and settings\username\my
documents\spirent\workspace" -t HTML -r "c:\\"
project://SpirentDemo/test_cases/testName.fftc"
```

Example 6

This example shows the arguments that an actual command might include (command prompt in blue text).

```
C:\Documents and Settings\[username]\My Documents\iTest_4.4_workspace\
my_project\test_cases> itestcli -p loop_count=5 -w "C:\Documents and
Settings\[username]\My Documents\iTest_4.4_workspace" -t HTML -r "C:\temp"
-es project://my_project/test_cases/favorite_testcase.fftc
```

Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

itestcli command-line arguments

Short form **Uses one hyphen character**	**Full text form** **Uses two hyphen characters and is identical operation to the short form**	**Description**

-a	**--about**	Display version information about the itestcli command line interface. Example response:
		`itestcli Command Processor Version 4.4.0`
		`Copyright (c) 2006-2010, Spirent Communications, Inc.`

-b {*URI* | *path*}	**--testbed** {*URI* | *path*}	Topology or testbed URI to use to execute the test cases
-c	**--connectToExistingServer**	Connect to the existing workspace server. If no workspace server is running, then quit.
-cf	**--configFolder** *path*	In some situations, you may want to use an existing iTest configuration (for example, you just upgraded and you want to continue to use your settings). Specify the fully qualified path to the Configuration folder to use for storing iTest configuration.
-d	**--debug**	Print diagnostic debug information
-es	**--summary**	Print a summary of test case execution
-expa *directory*	**--exportPath** *directory*	Specify the directory to export the itar files into. See -expr
--expr *projectName, projectName ..*	**--exportProject** *projectName, projectName ...*	Export the specified projects as itars, separated by commas. See -expa

-fip	**--forceIndeterminateAsPass**	Force Indeterminate result to return a Pass (**0**) return code. The default return code for an Indeterminate result (due to absence of an analysis rule) is Fail (**1**). If you specify the **-fip** option, then an Indeterminate result will instead return a Pass return code (**0**).
-g	**--guiserver**	Start a workspace server with a user interface - (required for web-based session types like Swing, Flex, and Web) Web, Flex, and Swing tests require a license for execution by itestcli. To use itestcli to run test cases that use Web and Swing sessions: Before running itestcli, run iTest and configure licensing. Any of the various iTest product edition license types will fulfill the requirement.
-h	**--help**	Display help for itestcli. (The contents of this **Command reference** section)
-i *time*	**--serverQuitIdleTime** *time*	Idle time in seconds after which the workspace server exits. Default, 60 seconds.
-l *path*	**--logfile** *path*	Output log file to collect execution information (path is a directory or file)
-n	--nologo	Do not display the logo
-p *parameter=value*	— None —	Optional. Override a parameter value. Repeat the **–p** option as often as needed to specify multiple parameter values. **Note** If you specify both **-p** and **-P** as sources for parameter values, then the values in the parameter file take precedence over the values that you specify using the **-p** *parameter=value* pairs.
-P URI	--param URI	During execution, use the parameters (and, if included, additional parameter files) specified in the parameter file identified by the URI. The URI can be for a parameter file in the workspace (relative to the test case URI) or for a file in the file system. See “Parameter files” on page 746 for details on creating a parameter file using the iTest user interface. **Note** If you specify both -p and -P as sources for parameter values, then the values in the parameter file take precedence over the values that you specify using the **-p** *parameter=value* pairs.

-qc	--publishToQualityCenter	Specify this option to publish the test report to Quality Center when execution finishes. Before executing the test case using itestcli, use the **Quality Center** page on the Test Case editor to publish the test case to Quality Center. This action sets the domain, project, and test instance location for the test case so that itestcli can auto-update the test instance with iTest test report data. See the **-qcp**, **-qcs**, and **-qcu** options
-qcd *domain*	**--qcDomain** *domain*	Optional. Specify the Quality Center domain. If the domain is not specified, then itestcli uses the value set in iTest preferences. To view or edit preferences, click **Window > Preferences** and then go to **Spirent > Quality Center**.
-qcp *password*	**--qcPassword** *password*	Required if you specify **-qc** (auto-publish the test report to Quality Center). Specify the password to connect to Quality Center.
-qcpr *project*	**--qcProject** *project*	Optional. Specify the Quality Center project. If the project is not specified, then itestcli uses the value set in iTest preferences. To view or edit preferences, click **Window > Preferences** and then go to **Spirent > Quality Center**.
-qcs *server*	**--qcServer** *server*	Optional. Specify the Quality Center server, overriding the workspace default If the server is not specified, then itestcli uses the value set in iTest preferences. To view or edit preferences, click **Window > Preferences** and then go to **Spirent > Quality Center**.
-qct *testInstance*	**--qcTestInstance** *testInstance*	Specify the full path to the test instance location in Quality Center format: "<Test Set Path>\<Test Instance Name>" For example, "Root\demo\mytestset\[1]mytestinstance"
-qcu *username*	**--qcUser** *username*	Required if you specify **-qc** (publish the test report to Quality Center). Specify the username to connect to Quality Center.
-r *path*	**--testreport** *path*	Directory name to store test report in or filename of the test report (path is a directory or file) See the **-sr** option.
-ra	**--reportAllSteps**	Include all executed steps in test reports (ignore the **Include this step and its children in test reports** property setting for the step).

-rw	**--refreshWorkspace**	Refresh workspace before executing because the workspace must be refreshed before Eclipse can locate a file that was added using file system tools outside of Eclipse, Use this option if, for example, you used the Windows Explorer to copy a test case file into the workspace directory,.
-sl	**-suppressLiveIssues**	Suppress printing of execution issues as they are generated (helpful to eliminate extra text at the terminal
-sr	**--subreports**	Generate test reports for this test case and for all child test cases. This option operates recursively (on all childrens' children, and so on). To use this option, you must also use the **-r** option to specify the path for the reports.
-ss	**--showSplash**	Display the iTest splash screen when starting up. You must ensure appropriate display settings.
-st *timeout*	**--serverStartTimeout** *timeout*	Time in seconds to wait to connect to the workspace server
-t *templateName*	**--testreporttype** *templateName*	Format of the test report file
-to *timeout*	**--timeout** *timeout*	Optional. Specify a timeout value in seconds to apply to all steps in all test cases in this invocation of itestcli. This setting overrides the **Default step timeout** property specified on the **General** page of the Test Case editor.
-trDbCatalog *name*	—	Optional. Specify the database/catalog name or SID. To connect to Oracle Database Express Edition, set the SID as **xe**. Default: reports
-trDbJdbcClass *class*	—	Optional. Java class for the custom JDBC driver for test report database. For example, **com.mysql.jdbc.Driver** See the topic on "Adding a custom third-party JDBC driver to iTest" in the *iTest Installation Guide*

trDb JdbcUrl *url*	—	Optional. URL of the JDBC connection for test report database. For example, **jdbc:mysql://[host][:port]/[database]** See the topic on "Adding a custom third-party JDBC driver to iTest" in the *iTest Installation Guide*. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
-trDbPassword *password*	—	Password to connect to the test report database
-trDbServer *ipaddr*	—	Hostname or IP address of the test report database server. Default: localhost
-trDbType *dbType*	—	Test report database type *dbType* can be: MySQL, SqlServer, Oracle, Sqlite, Postgresql, Derby, or Other
-trDbUser *userID*	—	userid to use to connect to the test report database.
-trGroup *workspace*	—	Optional. External database only. Test report group tag to report to the database If you specify this option, then: • The value acts as a parent to the optional **Subgroup** value. • Each report is tagged with the text. • You can use the value to search for test reports in the database. • The text appears in the **Group** column on the **Review Test Reports** activity page and on the Test Reports view.
-trHost *host*	—	Optional.Test report host tag to report to the database Default: iTest tags each test report with the hostname where the test case executes. If you specify text here, then each report is tagged with the text instead of the actual hostname. • You can use the value to search for test reports in the database. • The text appears in the **Host** column on the Review Test Reports activity page and on the Test Reports view.

-trSubGroup *project*	—	Optional. External database only. Test report subgroup tag to report to the database If you specify text here, then: • The value acts as a child of the **Group** value. • Each report is tagged with the text. • You can use the value to search for test reports in the database. • The text appears in the **Subgroup** column on the Review Test Reports activity page and on the Test Reports view.
-w *directory*	**--workspace** *directory*	Optional. Workspace to use for performing itestcli operations
-wm	**--wrappermode**	Print output from itestcli in a wrapper mode for parsing by scripts
-xml	**--xmlFormat**	By default, itestcli console output appears as plain text. Use the **-xml** option to format the output as XML. You can use standard operating system functions to redirect the output as needed (for example, to a file). See “Example XML console output (-xml option)” on page 703. Because output is printed to the console one line at a time, the XML will be incomplete if you abort execution. When multiple test cases are executed, itestcli displays the XML test case blocks one after the other. Any messages returned from the server are included in the output. Sample XML console output appears after this table. This option is not valid when used with the **-es**, **-sl**, or **-d**, arguments

Specifying an external database for test reports

You have the option to send all test reports to a JDBC-compatible database. There are two methods for specifying the database: auto-finding the database information and specifying the database information explicitly.

◆ Auto-find the database

The following arguments hide the complexity of specifying JDBC information for the database because iTest provides built-in support.

-trDbType <dbType>	Test report database type. Options: MySQL, SqlServer, Oracle, Sqlite, Postgresql, Derby, Other
-trDbServer <ipaddr>	Test report database server
-trDbPort <port>	Test report database port
-trDbCatalog <name>	Test report database catalog/name

-trDbUser <userid>	Test report database user ID
-trDbPassword <password>	Test report database password

- **Specify the database**

  The following arguments enable you to specify the JDBC-compatible database settings.

-trDbJdbcClass <class>	Test report database JDBC driver class
-trDbJdbcUrl <url>	Test report database JDBC connection string To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

Tagging test reports

You have the option to override the default database tags that are associated with test report database entries.

-trHost <host>	Default: Host where test case is stored
-trProject <project>	Default: Project where test case is stored
-trWorkspace <workspace>	Default: workspace where test case is stored

Example XML console output (-xml option)

```
<?xml version="1.0" encoding="UTF-8"?>
<itestcli version="4.4.n ">
   <testCase>
          <owner>userName</owner>
          <location>project://Web/TestCases/TestName.fftc</location>
          <startTime>03:09:13 AM</startTime>
          <executionIssue>
                 <severity>info</severity>
                 <origin>execution</origin>
                 <session/>
                 <step/>
                 <index/>
                 <procedure/>
                 <message>Execution started.</message>
          </executionIssue>
          <executionIssue>
                 <severity>Pass</severity>
                 <origin>exec.run</origin>
                 <session>s1</session>
                 <step>3</step>
                 <index>3</index>
                 <procedure>main</procedure>
                 <message>Test case "TestName" run result was "Pass" as
expected.</message>
```

```
            </executionIssue>
            <executionIssue>
                  <severity>info</severity>
                  <origin>execution</origin>
                  <session/>
                  <step/>
                  <index/>
                  <procedure/>
                  <message>Execution completed (4s)</message>
            </executionIssue>
            <name>AnchorTest</name>
           <issuesCount>4</issuesCount>
            <duration>00:00:04</duration>
            <finishTime>03:09:17 AM</finishTime>
            <result>Pass</result>
            <publishToServer>
                  <success>True</success>
                  <message>Success</message>
            </publishToServer>
            <message>Any message from iTest server</message>
            <testReport>/home/acme/results/
master_Fri_Sep_05_03_52_20_PDT_2008.html</testReport>

<testReport>/home/acme/results/submaster_Fri_Sep_05_03_52_20_PDT_2008.html
</testReport>
   </testCase>
     <testCase>
     <info>
               <error>Encountered an error executing
project://my_project/test_cases/testcase2.fftc</error>
               <reason>Missing -iu|--iUser argument</reason>
     </info>
     </testCase>


...
The <testCase> tag will repeat for multiple test cases passed in at the command
line
...

</itestcli>
```

Example of test case execution using itestcli

```
C:\Program Files\Spirent\iTest_4.4> itestcli -w "C:\Documents and
Settings\userName\workspace" "project://project/test_cases/MyTest.fftc"


itestcli Command Processor Version 4.4
Copyright (c) 2006-2012,Spirent Communications, Inc.


Executing testcase: 'project://project/test_cases/MyTest.fftc' ...
Execution started at: Wed Jun 10 13:59:44 PST 2010

Sev. Origin    Session  Step   Index  Procedure  Message
================================================================================
============================
Pass analysis  p1       2      1.4.2  mydut   Value '1' is consistent with '$
value == 1'
Pass execution                                    Test case MyTest has passed.

Execution finished at: Wed June 10 13:59:54 PST 2008
Execution Status: Pass

C:\Program Files\Spirent\iTest_4.4>
```

CHAPTER 32

iTest Runtime: iTestRT

In this chapter

Overview: iTestRT (iTest Runtime)

iTestRT is the command line (or “headless”) version of iTest that can operate on any iTest files whether or not they are held in a iTest workspace.

In headless mode (iTestRT) the **uriToPath** command extracts resources (file or directory) with the specified URI to a temporary directory and returns the target path (path of the extracted resources).

That is, the **uriToPath** in iTestRT, creates a temporary directory on the execution host and copies the contents of the URI in the command into the temporary directory. Each time the uriToPath command is executed in iTestRT, a new temporary directory will be created. Any temporary directories created during the test case are removed once the test case has completed.

Note Absolute paths cannot be used when executing the test cases from different systems (e.g., Windows-Linux).
You may use relative paths if **uriToPath** fails to translate “workspace locations” to absolute paths for external programs.
For example, if you execute test case in "**project://relativePath/test_cases/**" directory:
Copy files in the project using "**eval copy ../texts/msg ../texts/msgcopy**"

Note In iTestRT on Windows, displaying GBK characters (in listPrompts output) requires you to switch code page to 936. Use the following command before test execution: chcp 936.
Follow these steps to install new code pages:

- Press Windows+X to display the Power User menu and select the Control Panel.
- Open Region setting dialog and go to the Administrative tab.
- Click Change system locale and select Chinese (Simplified, China), click OK, and then reboot Windows.

Running iTestRT

To execute a iTest test case from the command line, you enter a command of the following form:

```
itestrt --test <testCaseFile>
```

Notes:

- Use `itestrt --help` for online help
- Progress is shown at the command line.

Example 1: Executing a test case

```
>itestrt --test project://my_project/demo/tc1.fftc

Executing test case: project://my_project/demo/tc1.fftc ...
Execution started at: Wed Apr 8 13:56:28 PST 2009 Test owner:

Sev.   Origin   Session  Step  Index  Procedure  Message
============================================================
info   execution                      Execution started
Pass   analysis       1    1    main     Response contains "Hello"
Pass   analysis       2    2    main     Response contains "World"
Pass   execution      2    2    main     Test case tc1 has passed.
info   execution                      Execution completed (0s)
Execution finished at: Wed Apr 8 13:56:29 PST 2009
Execution status:  Pass
```

Example 2: Executing a test suite

```
>itestrt --test project://my_project/sample_suite.ffts

Executing test suite: project://my_project/sample_suite.ffts ...

Result Group    Test       Start   Duration  Location
============================================================
Pass  group1   tc1.fftc     13:50:19 00:00:06
project://my_project/demo/tc1.fftc
Pass  group1   tc2.fftc     13:50:25 00:00:01
project://my_project/demo/tc2.fftc

Test Suite:           project://my_project/sample_suite.ffts
Start Time:           Wed Apr 8 13:50:19 PST 2009
End Time:            Wed Apr 8 13:50:27 PST 2009
Duration:             00:00:07
Completion Result:       Pass
Total tests executed:     2
Total tests passed:       2
Total tests failed/abort/ind: 0
```

Example 3: Running a iTest job to schedule execution

To run a job from the command line, enter a command of the following form:

```
itestrt --job [jobUri]
```

For full details, see “Running a job using iTestRT” on page 392.

Differences between iTestRT and itestcli

While both itestcli and iTestRT can be used for executing tests from the command line, there are some fundamental differences.

- Think of itestcli as a command line wrapper on top of iTest – when it boots, it loads up all the iTest modules, including all the GUI components used for test authoring which are then suppressed after being loaded into memory.
- In contrast, iTestRT is the much leaner iTest execution engine extracted from iTest, without any of iTest's test authoring and management capabilities. While iTestRT is limited in certain areas, iTestRT will eventually include all features and will completely replace itestcli.

HP ALM (formerly Quality Center) integration

iTest's ALM integration supports both iTestRT and itestcli for test execution and test report publishing to HP-ALM.

iTestRT command reference

All commands except iTestRT update commands are described in the following tables.

iTestRT update commands are described in "Updating iTestRT" on page 731.

Usage

```
itestrt [options]
```

Options and arguments are fully described in the following tables.

Note Some options include a URI as an argument. When specifying a URI, use a single slash character after "`file:`". For example:
`file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`

Option names

The option name listed in the table is the ***short form*** of the name. In some cases (fully automated regression, for example) you might want to use the full name. Using the full name for an option ensures that the correct option is used even if more than one module defines an option with the same short name (this can happen when someone develops and registers a new module using the **--options** option).

For example, the long form of the **projects** option names begin with:

com.fnfr.open.filesystem.itarproject.itestrtcmdline

Therefore the full name of the **--projects.list** option is:

--com.fnfr.open.filesystem.itarproject.itestrtcmdline.projects.list

Full names are identified in the table. To view full names for all registered options, use `itestrt --verbose`

Getting help

Use **itestrt --help** for a list of commands and their usage.

Use **itestrt --help <option ID or module ID>** for the description of an option.

Examples

--help paths

--help open.projects.paths

--help com.fnfr.open.projects.paths

--help com.fnfr.itest.testreport.optionModule

Syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[**x**]	Keywords or arguments that appear within square brackets are optional.
{**x** | **y** | **z**}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.

Checking out a runtime license

To execute a test case, test suite, or job, iTestRT must obtain a runtime license by providing the license server address and port number.

--licenseServer *hostAddress:portNumber*	*hostAddress* is the hostname or IP address of the license server host (typically provided by your IT administrator). iTestRT uses port **27000** if you do not specify a port number. To use more than one license server, use the **--licenseServer** argument multiple times at the command line. **Tip** Set up multiple license servers in the options file. **Example** >iTestRT --licenseServer lshost.acme.com:-1 --test project://my_project/test_cases/test1.fftc

NTAF automation: Running test cases that include NTAF sessions

To execute an NTAF test case using iTestRT, first start the Spirent NTAF proxy, and then connect to the NTAF server using the NTAF options. For example:

```
iTestRt --itar file:/c:/iTestRt --test
project://my_project/TestCases/NtafAvTest42.fftc --ntaf.server crt-fm5q1
--login itestrt --ntaf.password mypassword
```

--**domain** *URI*	XMPP domain name. Default: The NTAF server hostname. When you log onto an XMPP server, you get an ID (called a Jabber ID, JID) like username@ntafxmpp.spirent.com/unspecified “username” is the user, "ntafxmpp.spirent.com" is the domain, which is usually the same as the value of the **NTAF server**. However XMPP servers can be configured so that domain and server are different. (In the example, “unspecified” is the **Resource**). **Note** Use a single slash character after “`file:`” in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>` **Note**
--inband *value*	In-band registration means that the NTAF server will register a new account for your username if it does not yet exist. **Note** The logic of the following *values* seems reversed, but the following descriptions are correct: **false** — Try in-band registration if login fails. Your username must exist on the NTAF server for this option. **true** — Do not try in-band registration Default behavior: **false** — try in-band registration if login fails.
--**login** *name*	XMPP credential for the NTAF server. Default: The value of the **Username** credential used to log in.
--**ntaf.port** *value*	Port address of NTAF server. Default: 5222
--ntaf.server *URI*	Specify the IP address or hostname of the NTAF server (provided by your IT administrator) **Note** Use a single slash character after “`file:`” in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--ntaf.user *value* **--ntaf.password** *value*	Specify the XMPP username/password credentials to use to enable iTest to access the NTAF server as a client. **Note** The credentials represent iTest as a client on the NTAF server. Remember that the Proxy service is a different client on the NTAF server and therefore has a different username. You can set the authentication information in the **NTAF** preferences page in iTest. The values are not used when an application login page asks for credentials
--**reconnect** *value*	**Note** The logic of the following *values* seems reversed, but the following descriptions are correct: **false** — Retry connecting if the connection with the XMPP server fails **true** — Do not retry Default behavior: **false** —retry when connection fails

--regAddress *value*	Pubsub server address on NTAF server. Default: “pubsub.” followed by the XMPP domain.
--regRoot *value*	Pubsub root node for the NTAF registry. All NTAF provider registration information is stored under the root node. Default: **ntaf.tools** The default setting is typically correct. While the NTAF standard allows for other names, changing names may confuse the providers that you are trying to communicate with.
--**resource** *value*	XMPP resource. The last part of the JID is the resource. For example, “unspecified” in **username@ntafxmpp.spirent.com/unspecified** Default: “unspecified”

Workspace, Projects and options

The **projects** options support itar files that are specified at the command line. In addition, iTestRT **--itar** command supports executing test cases from iTest workspace when you specify Workspace as the **itar** path. This also enables execution of all test case commands in iTestRT within the workspace context and query any workspace related information (e.g., info workspacePath). However, it is *not* recommended to use the `[info workspacePath]` to determine if iTest is running in headless mode.

The long forms of option names begin with: **com.fnfr.open.filesystem.itarproject.itestrtcmdline.**

--itar *URI*	Specifies the URI of the directory that contains the itar files. (one directory per **--itar** option) **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--itar *command*	Specify Workspace folder as **itar** path, to run any test case contained in the expanded project folder or a packed itar, using the **--test** option. **Example iTest Explorer** **RTWORKSPACE** **itar_test** **test_cases** CustSessionTC **misc_folder** **...** **my_project** **test_cases** testcase.fftc **misc_folder** **...** **Example 1: Run test case in folder "my_project"** **--itar** `c:\users\user_name\RTWORKSPACE --test project://my_project/test_cases/testcse.fftc --licenseserver itest-lic.mycompany.local` **Example 2 Run test case in folder "itar_test"** **--itar** `c:\users\user_name\RTWORKSPACE --test project://itar_test/test_cases/CustSessionTC.fftc --licenseserver itest-lic.mycompany.local` **Note** If a Workspace contains 2 projects with the same name, a packed **itar** file and the other expanded project folder, then the test case contained in the expanded project folder will have higher priority for test execution. For example, if iTest workspace called RTWORKSPACE, contains 2 projects, **itar_test .itar** and **itar_test**, the expanded project folder, then the test case **CustSessionTC.fftc**, contained in the **itar_test** folder will have higher priority for test execution.
--paths	Lists the paths that are searched for itar files.

--projects.list	Lists projects that are available
--exportItar	Use **--exportItar** command to export iTest projects. The example command below shows the workspace to used to create iTAR file using the **--itar** command exporting it using the **--exportItar** option. **--itar** `c:\users\user_name\RTWORKSPACE --licenseserver itest-lic.mycompany.local` **--exportItar** An **iTar** is created for every project in your workspace and resources are also treated as a project. You may execute test case from these iTar files use the command shown below. See also "--itar command" on page 713. **--itar** `c:\users\user_name\RTWORKSPACE --licenseserver itest-lic.mycompany.local` **--test** `project://itar_test/test_cases/my_testcase.fftc` **Note** The **--exportItar** command only exports projects and it does not build or validate iTar files.

Agent option: Running iTestRT as an agent

The **Velocity** scheduling and execution service uses one or more instances of iTestRT running in agent mode.

The long forms of option names begin with: **com.fnfr.open.runtime.xmpp.agent.optionModule**

To run iTestRT as an agent

Execute **iTestRT** using the following arguments:

--licenseServer	Required. See "Checking out a runtime license" on page 710.
--agent	Run iTestRT as an agent
--icuser *username*	The agent should log into the server using the specified username.
--icpassword *password*	The password for the specified user
--icserver *URI*	URL of an **Velocity** instance, for example, http://somehost:8080 To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html** **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`

Note Use the --chartcolor option to specify the color palette to be used when generating/print charts.

For example:

```
--agentVelocityHost 100.100.10.8
--report file://D:/workspace/ChartColor/result/ --chartcolor iTest
```

Job options

The **job** options support scheduled execution by running iTest jobs.

The long forms of option names begin with: **com.fnfr.open.runtime.jobexecutor.optionModule**

--licenseServer	Required. See "Checking out a runtime license" on page 710.
--job *URI*	Specifies the URI of the job file to run **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`

--job.logfile *URI*	Specifies the URI of an output log file that will collect execution information. You must specify the log file before the associated **--job** *URI* option. The **--job.logfile** option applies to all subsequent **--job** *URI* options, unless the **logfile** option is overridden by a **logfile** option with a new value. That is, if a command includes multiple instances of the of the **--job.logfile** option for an associated **--job** *URI* option, then only the last **logfile** instance is used. **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--job.quiet	Suppresses output from the job run

File Utilities options

The 'file utilities' options enable you to list files in projects.

The long forms of option names begin with: **com.fnfr.open.runtime.fileutils.optionModule**

--cat *URI*	Prints the contents of the specified URI **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--fileutils.list *URI*	Prints a list of the files for the specified URI **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--recursive	Causes the list of files to be recursive

Test Execution options

The test execution options support executing a test case, test suite, or job.

Required: Check out a runtime license

See "Checking out a runtime license" on page 710.

Execution options

You must specify all test execution options before the associated **--test** *URI* option.

Note For each test execution option (except **--quiet**): When multiple instances of the option appear before a particular instance of **--test**, then only the last instance is used. In this example, `b.log` will be used for `job1` execution and `c.log` will be used for `job2` execution:

```
itestrt --licenseServer lshost.acme.com:-1 --log file:/C:/a.log --log
file:/C:/b.log --test file:/C:/job1.ffjd  --log file:/C:/c.log  --test
file:/C:/job2.ffjd
```

The long forms of option names begin with: **com.fnfr.open.runtime.executionengine**

--licenseServer	Required. See "Checking out a runtime license" on page 710.
--duration *duration*	Time period required for the reservation
--fip	Optional. Force Indeterminate result to return a Pass (**0**) return code. The default return code for an Indeterminate result (due to absence of an analysis rule) is Fail (**1**). If you specify the **-fip** option, then an Indeterminate result will instead return a Pass return code (**0**)
--param *parameter=value*	Optional. Specifies a parameter value for the test to override a parameter value. Repeat the **–-param** option as often as needed to specify multiple parameter values. **Note** If you specify both **--param** and **--paramfile** in a iTestRT command, then values that you specify using the **--param** argument take precedence over the values in the parameter file.

--paramfile *URI*	Optional. During execution, use the parameters (and, if included, additional parameter files) specified in the parameter file identified by the URI.
	iTestRT also supports using query language within parameters, which are resolved to the specified resource property value when running the test case.
	See "Parameter files" on page 746 and "Working with parameters: The Parameters page" on page 733 for details of using the iTest user interface for creating a parameter file and also using the Property Query Language syntax.
	Note Use a single slash character after "`file:`" in the URI. For example:
	`file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
	For example:
	`--velocityServer https://<host>/velocity --velocityLogin <login> --velocityPassword <password> --itar <location> --paramfile project://DynamicParameters/DynamicParameters.ffpt --test project://DynamicParameters/DynamicParametersPC.fftc`
	Note If you specify both **--param** and **--paramfile** in a iTestRT command, then values that you specify using the **--param** argument take precedence over the values in the parameter file.

--reportallsteps	Include all executed steps in test reports (ignore the **Include this step and its children in test reports** property setting for the step).

--test *URI*	Specifies the URI of the test case or test suite to be executed. **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>` **Note**
--test.logfile *URI*	Optional. Specifies the URI of an output log file that will collect execution information. **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>` **Note**
--test.quiet	Optional. Suppress output during execution.
--testbed *URI*	Optional. Specifies the URI of the testbed or topology to use for execution. **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>` **Note**
--timeout *seconds*	Optional. Execution timeout for the test specified by the **--test** option.

Velocity integration

To run a test case associated with a Velocity topology (using the **ilo** command to retrieve information about the active topology), specify the following parameters:

---iloLogin *userName* **---iloPassword** *password*	Specify the username/password credentials to use to access the **Velocity** server
--iloServer *URI* **/ilo**	*URI* is the hostname or IP address of the **Velocity** virtual appliance. The URI is followed by "**/velocity**" **Example** `--velocityServer http://velocity.acme.com/ilo`
--reservationId *reservationId*	If there is more than one active reservation for the topology associated with the test case, then you must specify a value for the **--reservationId** option. When there is only one active reservation of the topology, then you do not need to specify a value for **--reservationId**

Example:

```
itestrt --licenseServer lshost.acme.com:27000 --iloServer
http://ilo.acme.com/ilo --iloLogin Apurba --iloPassword yikes --reservationId
1e4371d0-e8f2-4ecb-91e2-b1e67535b867 --itar C:\itars --test
project://my_project/test_cases/telnet3.fftc
```

Launcher options

The launcher options enable you to control iTestRT startup.

--help [*optionName*]	Prints help information
--options *path*	Load additional (non-built-in) options from the file specified by *path.* *path* is a directory or file.
--verbose	Prints verbose help that identifies the full name of each registered option.

Test Report options

The **report** options enable you to control how test reports are generated. The long forms of option names begin with: **com.fnfr.itest.testreport.optionModule**

On the test report's **Response** section, iTest displays response format as **JSON** or **Text** form (as auto-detected).

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.

  Only text indentation is applied (to HTML, HTML/JSON, PDF, and customized reports) and not the text color.

- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in , "JSON Editor").

--report *URI*	Generate test report at the specified URI. **Note** If a command includes multiple instances of the of the **--report** option, then only the last instance is used. The following substitutions are supported in the text of the URI: {tcfilename}, {datetime}, {date}, {time}, {tcpath} (test case path) {jobfilename}, {jobstart} **Note** You must append an appropriate file extension to the end of the replacement text, for example, **{tcfilename}.html** **{tcfilename}.xml** **{tcfilename}.xml_raw** **{tcfilename}.txt** The report format will be determined by the specified file extension and iTest supports 5 types of format: HTML, Text, XML, XML_Raw and PDF. The file extension supported is predefined in file extension.txt with appropriate text (html, xml, xml_raw, or txt) and located in test_report_templates/format_name/. If the extension appended does not match any predefined format, HTML format will be used, the file name with extension are kept as provided, and any existing XSLT stylesheet is applied. In addition, the test report may be compressed when publishing to the Quality Center server. **Note** For Microsoft Windows 7: You must run the CLI as “Administrator' to use the **--report** option to store the report in a folder. (If not, then an “Access Denied” error occurs) **Example** **--report project://my_project/stressTests/{tcfilename}{datetime}.html** **Note** Use a single slash character after “`file:`” in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--quiet	Suppress output from test report generation. **Note** Not used in conjunction with the **--comparereports** option.

--format *URI*	Specifies the templates to use to format HTML, PDF, TEXT, XML, or XML_RAW reports. If you use this option, then you do not need to export the **resources** project to an itar file. To use the option: 1. Copy the **reportsXX/test_report_templates/**<format_name> folder from your **resources** project into a local folder. 2. Specify the URI of the new folder using the **--format** option. **Example** 1. Copy **project://resources/reportsXX/test_report_templates/HTML** to **C:\templates\my_templates** You may copy the custom templates and specifiy the URI during execution. 2. Now, when executing iTestrRT, use**--format file:/C:/templates/my_templates** **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`
--report *URI* **?***format*	Generate test report at the specified URI in the specified format. Example: **--report file://C:/?XML_Raw** The format output will be xml and the report name will be **testcasename+timestamp.xml.** **--report file://C:/abc?XML_Raw** The format output will be xml and the report name will be **abc.testnumber.xml**. **Note** The report format will be determined by the specified file extension and iTest supports 5 types of format: HTML, Text, XML, XML_Raw and PDF. The file extension supported is predefined in file extension.txt with appropriate text (html, xml, xml_raw, or txt) and located in test_report_templates/format_name/. If the extension appended does not match any predefined format, HTML format will be used, the file name with extension are kept as provided, and any existing XSLT stylesheet is applied. In addition, the test report may be compressed when publishing to the Quality Center server. iTestRT auto-detects JSON response data, and if the response is valid JSON, iTest formats the response as JSON prettiy print. **Note** The above applies to both test case and test suite.

--chartcolor <*ChartColor*>	Specify the color palette to be used to generate charts. if no chart color is specified, the color palette iTest is used by defaul.
	You can use the **itestrt --help** to list the available color pallette:
	`Chart Color Options:`
	`--chartcolor <chartColor>  Specify color for chart's series <iTest, Beach,Bermuda, Fall, gentleman, marine, Party, Playground, Vacation, Velocity>`
	Example usage:
	`--itar file://D://workspace/ChartColor/ --test project://my_project/test_cases/ new_chart_testCase.fftc --licenseServer 100.100.10.1 --report file://D:/workspace/ChartColor/result` **`--chartcolor`** `iTest`

Test report database options

The **Rest report database** enable you to set up response compression and save test reports to a specified database (instead of the built-in iTest database). See Chapter 21, “Test Reports”, “Editors > Test Reports > Database” on page 457.

Important Database settings that you make using iTestRT remain in use until you change them using iTestRT.

The long forms of option names begin with: **com.fnfr.open.runtime.test.trdb**

--catalog *name*	Name of the database or catalog.
--configonly	Configure database and ignore test options.
--dbtype *type*	Database type. Allowed values: MySQL, SqlServer, Oracle, Sqlite, Postgresql, Derby, Other. If **Other**, you must specify a value for the **driverclass** argument (JDBC connection string and class).
--driverclass *class*	Java class name of the JDBC driver for the database. Used when **dbtype** is set to **Other**.
--disableResponseCompression	By default responses are compressed in test reports database. Use this option to indicate that the database should not be compressed.
--group *tagText*	Group tag to associate with the report. **Note** Not used in conjunction with the **--comparereports** option.
--host *tagText*	Host tag to associate with the report. **Note** Not used in conjunction with the **--comparereports** option.
--ipaddr *IpAddress*	IP address of the database server.
--port *portNumber*	TCP port of the database server.
--project *tagText*	Project tag to associate with the report. **Note** Not used in conjunction with the **--comparereports** option.

--subgroup *tagText*	Subgroup tag to associate with the report. **Note** Not used in conjunction with the **--comparereports** option.
--tag *TagName=tagText*	This option enables you to define and assign a value to a custom tag. **Example** Create a tag that holds the build number so that you compare test execution results between builds. For tests run against build **54321**, use: **--tag buildNumber=54321** For tests run against build **54322**, use: **--tag buildNumber=54322** **Tip** Use a custom tag to identify executions or groups of executions on the **Velocity** Test Execution page.
--trdb.password *password*	Password for logging in to the database server.
--trdb.user *user*	User ID for logging in to the database server.
--responseCompressionThreshold <size>	Allows you to make a test report database more compact. (Normally, test case responses occupy a lot of space in a test report database, internal or external). By default, the response compression is disable in iTestRT. Enter an integer value to indicate the size in bytes. Responses with size greater than the specified value (argument) will be compressed in the test reports database. Simple integer value is interpreted as size in bytes. You may also specify size in kilobytes or megabytes **Example** 100KB or "100 KB" or 5MB or "5 MB" **Note** The quotation marks usage helps avoid command line parsing error due to spaces.
--uri *URI*	Connection URI to use to connect to the database. **Note** Use a single slash character after "`file:`" in the URI. For example: `file:/C:/Workspace/my_project/<folder>/<filename>.<extension>` **Note**

Test Report Comparison

The **reportcomparison.comparereports** option compares two test reports and generates a "diff" report that shows the differences between the two reports. More details on comparing reports appears in "Comparing ("diffing") two test reports" on page 444.

The **comparereports** command uses the following options:

- Test Report options specify report format and location (see "Test Report options" on page 719)
- Test report database options configure the report database options (see "Test report database options" on page 722)

Note You must specify the **--configonly** database option to configure the test report database properly for this option. See "Test report database options" on page 722.

The long forms of option names begin with: **--com.fnfr.open.runtime.reportcomparison.**

--comparereports *SourceID*,*TargetID*	SourceID and TargetID are the comma-separated report IDs of source and target reports that should be compared.

Test Report Publishing Service (rps) options

The **rps** options enable you to publish reports to a service.

Long form of name begins with: **com.fnfr.open.automation.rps.activator**

--rps.password *password*	Account password for logging in
--rps.user *userName*	Account user name for logging in
--server *serverURL*	URL of the server to publish to To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

XMPP

The **xmpp** options enable you to set XMPP connection advisor parameters at the command line.

The long forms of option names begin with: **com.fnfr.itest.runtime.xmpp.advisor.optionModule**

--xmpplogin *loginName*	Login name
--xmppuser *userName*	User name
--xmppfeature *feature*	Feature
--xmppinband	Register if does not exist
--xmpppub *ipAddr*	IP address of the pubsub
--xmppserver *server*	Host
--xmpppassword *password*	Password
--xmppresource *resource*	Resource
--xmppservice *serviceName*	Name of the XMPP server to connect to
---xmppport *port*	Port where the XMPP is listening

Launching a iTest Rational Quality Manager adapter using iTestRT

You start RQM adapters from iTestRT. See "Launching a iTest RQM adapter" on page 941 for details.

Quality Center options for iTestRT

The **qc** options cause iTest to publish the test report to the Quality Center server, domain, and project that you specify at the command line.

Important The command line arguments has higher priority and overrides the Test Case parameters.

The long forms of option names begin with: **com.fnfr.svt.features.integration.hpqc**

--qc	Auto-publish the test report to Quality Center when execution finishes. **Note** The report format will be determined by the specified file extension and iTest supports 5 types of format: HTML, Text, XML, XML_Raw and PDF. The file extension supported is predefined in file extension.txt with appropriate text (html, xml, xml_raw, or txt) and located in test_report_templates/format_name/. If the extension appended does not match any predefined format, HTML format will be used, the file name with extension are kept as provided, and any existing XSLT stylesheet is applied. In addition, the test report may be compressed when publishing to the Quality Center server. **Note** If you specify --qc then the following args are required: **--qcd**, **-qcp**, **--qcpr**, **--qcs**, **--qcu, qcrf,** and **qcZip**
--qcd *domain*	Required if you specify **--qc**. Specify the Quality Center domain.
--qcp *password*	Required if you specify **--qc**. Specify the password to connect to Quality Center.
--qcpr *project*	Required if you specify **--qc**. Specify the Quality Center project.
--qcs *server*	Required if you specify **--qc**. Specify the Quality Center server.
--qct *testInstance*	Optional. If you specify a QC test instance, then iTestRT uses the information in the test instance instead of the information in the iTest test case to determine where to write the report. If you specify a test instance, then you can specify only one test case for execution (you cannot specify a job or test suite). If you do not specify a test instance, then iTestRT uses the information in the iTest test case Specify the full path to the test instance location in Quality Center format: "<Test Set Path>\<Test Instance Name>" For example, "Root\demo\mytestset\[1]mytestinstance"
--qcu *username*	Required if you specify **--qc**. Specify the username to connect to Quality Center.

--qcrf *ReportFormat*	Optional.
	When the option is specified, the report format uploads to QC server as specified in the format **--qcrf**.
	For example:
	If you would like to have two different report formats, one saved locally and the other on the Server, use the command as follows.
	Both these URI formats are supported on iTestUI and RT:
	project:// and **file:/**
	`--licenseServer license-host --test project://my_project/test_cases/test_qc.fftc`
	`--qcrf project://resources/reports24/test_report_templates/XML --qc --report file://D:/reports/ --format project://resources/reports24/test_report_templates/Text`
	• The XML report will be uploaded to QC Server.
	• The text report will be saved locally to: D:/reports/.
	Note If **--qcrf** is not specified, report format uploaded to server will be the same as the saved local report.

--qcZip	Optional. When specified, the **qcZip** option compresses the test report when publishing to the Quality Center server. For example: `--licenseServer xxx.xxx.xx.xxx --parameter project://my_project/session_profiles/hpqcparam.ffpt --itar c:\itar --test project://my_project/test_cases/publish_zip_test_case.fftc --qczip` **Note** The report format compressed depends on the option specified (--qcrf). If not specified, the compressed report format uploaded to server will be the same as the saved local report.
--qcSet	Use paramter file to auto fill customized fields for {Test, Test Instance, Test Run} when publishing from iTest. The following lists the precedence of paramater file and test case paramters used witht he **qcSet** option: • iTest RT: RT option > Param file> Test case param • Agent: Param file > Test case param > agent option **Example:** `--licenseServer xxx.xxx.xx.xxx --qc --itar file:/D:/itarFolder --paramfile project://my_project/session_profiles/param.ffpt --test project://my_project/test_cases/demo_autofill_testcase.fftc --qcSet "{'Test': [{'USER-01': 'NEW_TS_THANG_01'},{'USER-02': 'NEW_TS_THANG_02'}],'Instance': [{'USER-01': 'NEW_TC_THANG_01'},{'USER-02': 'NEW_TC_THANG_02'}],'Run': [{'USER-01': 'NEW_RN_THANG_01'}]}"`

Examples: Quality Center options for iTestRT

Example 1

```
>iTestRT.bat --itar F:/iTestRT_workspace --test
project://test_cases/DiffArtifactsFolder.fftc --qcu manual --qc --qcp manual --qcpr
MANUALTEST --qcd Spirent --qcs http://qcfast/qcbin

Executing test case: project://test_cases/FR_DiffArtifactsFolder.fftc ...
Execution started at: Wed Jun 2 10:58:32 PDT 2010
Test owner: ronakdhar

Sev.   Origin   Session  Step  Index  Procedure  Message
============================================================
Info   execution              main    Execution started
Pass   analysis       3   3    main    Query "group4()" matched value "ArtifactsFolder2",
which is equal to "ArtifactsFolder2"
Pass   analysis  t1    6   6    main    Response contains "Cisco"
Pass   execution  t1    6   6    main    Test case FR_DiffArtifactsFolder has passed.
Info   execution                     Execution completed (2s)
Execution finished at: Wed Jun 2 10:58:35 PDT 2010

Execution status: Pass
Report Id: 1
Test results for FR_DiffArtifactsFolder.fftc successfully uploaded to Quality Center.
```

Example 2

```
>iTestRT.bat --itar F:/iTestRT_workspace --test
project://test_cases/DiffArtifactsFolder.fftc --qcu manual --qc --qcp manual --qcpr
MANUALTEST --qcd Spirent --qcs http://qcfast/qcbin
 Executing test case: project://test_cases/DiffArtifactsFolder.fftc ...
Execution started at: Wed Jun 2 11:02:45 PDT 2010
Test owner: ronak

Sev.   Origin   Session  Step  Index  Procedure  Message
============================================================
Info   execution              main    Execution started
Warning  message        1.1.1 1.1.1  main    No workspace, just stop
Pass   analysis       1.1.1 1.1.1  main    Pass anyway
Pass   execution      1.1.1 1.1.1  main    Test case DiffArtifactsFolder has passed.
Info   execution              main    Execution completed (1s)
Execution finished at: Wed Jun 2 11:02:47 PDT 2010

Execution status: Pass
Report Id: 1
```

Example 3

```
>iTestRT.bat iTestRT.bat --itar F:/iTestRT_workspace --test
project://test_cases/multiple_runs.fftc --qcu manual --qc --qcp manual --qcpr MANUALTEST
--qcd Spirent --qcs http://qcfast/qcbin

 Executing test case: project://test_cases/multiple_runs.fftc ...
Execution started at: Wed Jun 2 11:04:13 PDT 2010
Test owner:

Sev.   Origin   Session  Step  Index  Procedure  Message
============================================================
Info   execution               main     Execution started
Pass   exec.run       1   1    main    Test case "DiffArtifactsFolder" run result was "Pass"
as expected.
Pass   exec.run        2   2    main     Test case "FR_DiffArtifactsFolder" run result was
"Pass" as expected.
Pass   execution       2   2    main     Test case multiple_runs has passed.
Info   execution                       Execution completed (5s)

Test Summary:
Completion Result: Pass
Test Case:      multiple_runs
Location:       project://test_cases/multiple_runs.fftc
Owner:
Start Time:      Wed Jun 02 11:04:13 PDT 2010
Duration:       00:00:06
Total Issues:    5
Report Id:      1
Child Test Cases : 2 with 0 unexpected results

* - indicates unexpected result


No child test cases with unexpected results


Child Test Cases (All Results) : 2
* Result Testcase          Owner    Duration StartTime  ReportId Location
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
 Pass  DiffArtifactsFolder   ronak 00:00:00 11:04:15 AM 10
project://test_cases/DiffArtifactsFolder.fftc
 Pass  FR_DiffArtifactsFolder ronak 00:00:02 11:04:16 AM 19
project://test_cases/FR_DiffArtifactsFolder.fftc



Execution issues for this test case:
Sev. Proc. Step Message
-------------------------------------------------------------------------------------------
-------------------------------------------------------------
info main     Execution started
pass main  1   Test case "DiffArtifactsFolder" run result was "Pass" as expected.
pass main  2   Test case "FR_DiffArtifactsFolder" run result was "Pass" as expected.
pass main  2   Test case multiple_runs has passed.
info        Execution completed (5s)

Execution finished at: Wed Jun 2 11:04:19 PDT 2010

Execution status:  Pass
Report Id:  1
```

Example 4

```
>itestrt.bat --itar F:/iTestRT_workspace --test project://test_cases/qctest_suite.ffts
--qcumanual --qc --qcp manual --qcpr MANUALTEST --qcd Spirent --qcs http://qcfast/qcbin

 Executing test suite: project://test_cases/qcte
st_suite.ffts ...

Result Owner    Group       Test        Start   Duration  ReportId
 Location
================================================================================
=======
Pass  ronak group1       BlankNameQCArtif 11:25:22 00:00:13  1
 project://test_cases/BlankNameQCArtifactsFolde
r.fftc
Pass  ronak group1       DiffArtifactsFol 11:25:35 00:00:00  10
 project://test_cases/DiffArtifactsFolder.fftc
Pass  ronak group1       EventInherit_qcS 11:25:36 00:00:00  19
 project://test_cases/EventInherit_qcSet.fftc
Pass  ronak group1       FR_DiffArtifacts 11:25:36 00:00:02  28
 project://test_cases/FR_DiffArtifactsFolder.ff
tc
Pass  ronak group1       InvalidQCArtifac 11:25:38 00:00:00  37
 project://test_cases/InvalidQCArtifactFolder.f
ftc
Pass         group1       multiple_runs.ff 11:25:39 00:00:01  46
 project://test_cases/multiple_runs.fftc
Pass  ronak group1       ReadOnlyArtifact 11:25:41 00:00:00  73
 project://test_cases/ReadOnlyArtifactFolder.ff
tc
Pass  ronak group1       Workspace_Artifa 11:25:41 00:00:00  82
 project://test_cases/Workspace_ArtifactsFolder
.fftc

Test Suite:            project://test_ca
ses/qctest_suite.ffts
Start Time:            Fri Jun 4 11:25:22 PDT 2010
End Time:              Fri Jun 4 11:25:42 PDT 2010
Duration:              00:00:19
Completion Result:        Pass
Total tests executed:      8
Total tests passed:       8
Total tests failed/abort/ind: 0
```

Example 5

```
>itestrt.bat --itar F:/iTestRT_workspace --test project://test_cases/qctest_suite.ffts
--qcu manual --qc --qcp manual --qcpr MANUALTEST --qcd Spirent --qcs http://qcfast/qcbin
--qct test1

 Error during initialization: When using the qct option, you can specify only one
            test case for execution
```

Example 6

```
>itestrt.bat --itar F:/iTestRT_workspace --job
project://test_cases/qc_testjob.ffjd --qcu manual --qc --qcp manual --qcpr MANUALTEST --qcd
Spirent --qcs http://qcfast/qcbin

 1. Loading job 'project://test_cases/qc_testjob
.ffjd' - OK
   qc_testjob (Once immediately)

Running jobs:
<None>

Scheduled jobs:
#  Job Details             Next Start Time
------------------------------------------------------------------------------
----------------------------------------------------
1  qc_testjob (Once immediately)  11:40 AM Jun 4, 2010




Starting: qc_testjob (Once immediately)
Running jobs:
#  Job Details             Start Time        Elapsed Time
------------------------------------------------------------------------------
----------------------------------------------------
1  qc_testjob (Once immediately)  11:40 AM Jun 4, 2010  0s


Scheduled jobs:
<None>



Executing test suite: project://test_cases/qcte
st_suite.ffts ...

Result Owner    Group       Test        Start   Duration  ReportId
 Location
==============================================================================
=======
Pass  ronak group1       BlankNameQCArtif 11:40:37 00:00:13  1
 project://test_cases/BlankNameQCArtifactsFolde
r.fftc
Pass  ronak group1       DiffArtifactsFol 11:40:50 00:00:00  10
 project://test_cases/DiffArtifactsFolder.fftc
Pass  ronak group1       EventInherit_qcS 11:40:51 00:00:00  19
 project://test_cases/EventInherit_qcSet.fftc
Pass  ronak group1       FR_DiffArtifacts 11:40:52 00:00:01  28
 project://test_cases/FR_DiffArtifactsFolder.ff
tc
Pass  ronak group1       InvalidQCArtifac 11:40:53 00:00:00  37
 project://test_cases/InvalidQCArtifactFolder.f
ftc
Pass         group1       multiple_runs.ff 11:40:54 00:00:02  46
 project://test_cases/multiple_runs.fftc
Pass  ronak group1       ReadOnlyArtifact 11:40:56 00:00:00  73
 project://test_cases/ReadOnlyArtifactFolder.ff
tc
Pass  ronak group1       Workspace_Artifa 11:40:56 00:00:00  82
 project://test_cases/Workspace_ArtifactsFolder
```

```
.fftc

Test Suite:            project://test_ca
ses/qctest_suite.ffts
Start Time:            Fri Jun 4 11:40:37 PDT 2010
End Time:             Fri Jun 4 11:40:57 PDT 2010
Duration:             00:00:19
Completion Result:        Pass
Total tests executed:      8
Total tests passed:       8
Total tests failed/abort/ind: 0
Finished: qc_testjob (Once immediately)
   Test Result: Executed [PASS]
   Run status: OK
Running jobs:
<None>

Scheduled jobs:
<None>


Test results for FR_DiffArtifactsFolder.fftc successfully uploaded to Quality Ce
nter to artifact_test\[3]artifact_test.
Test results for FR_DiffArtifactsFolder.fftc successfully uploaded to Quality Ce
nter to artifact_test\[3]artifact_test.
Test results for multiple_runs.fftc successfully uploaded to Quality Center to R
oot\ronak\multiple_test\[1]multiple_test.
```

Running test cases that are associated with Velocity topologies

If you do not specify **reservationId** as an argument to iTestRT, and only single reservation of the associated topology owned by requesting user exists, iTestRT will assume that reservation.

If you do not specify **reservationId** as an argument of iTestRT, and either no reservations exists or more than one reservation of the associated topology owned by the requesting user exists, it is an error.

Updating iTestRT

You can update all currently installed iTestRT features from the command line. There are options for:

- Updating only particular features
- Verifying that a particular update process will proceed

See the *iTest Installation Guide* chapter for detailed instructions.

CHAPTER 33

Parameters

You can use parameters to provide values that a test case should use as it executes.

At runtime, the test case can use parameter values from several sources: testbeds, session profiles, and the parameters that you configured on the test case's **Parameters** page. You can also specify a Global parameter file so that any executing test case uses the parameters defined in the file. In addition, if the Global parameter file references other parameter files, then the parameter definitions in the referenced files are included.

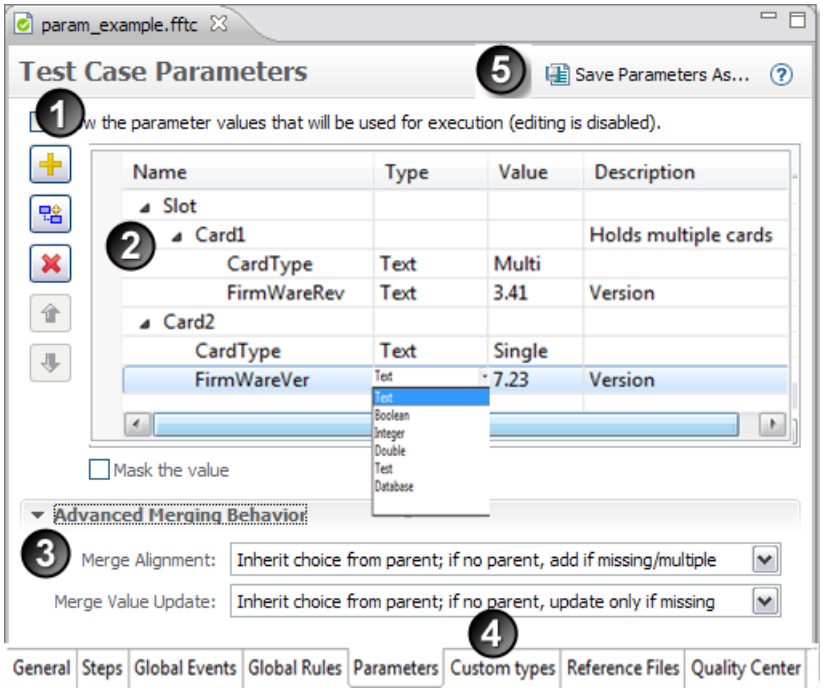
Because you can combine parameter settings from so many locations, you can implement quite sophisticated dynamic operation.

Defining and managing parameters

Working with parameters: The Parameters page

The **Parameters** page, where you define parameters, appears on the Test Case editor, the Testbed editor, the Session Profile editor, and the Parameter editor.

Important The **Parameters** page that appears on the Parameter editor differs from the **Parameters** page on the other editors in the following way: While you work in the Parameter editor, you are editing a ***parameter file***. While you work in one of the other editors, you are editing individual parameter definitions for the current document (test case, testbed, or session profile). See "Parameter files" on page 746.



On the **Parameters** page, you can:

1 ***Define and edit parameters***

Use the toolbar to add a parameter and then specify its **Name** and **Value**, and optionally, a **Description** that will help your coworkers to understand the usage of the parameter.

In the example, the **firmwareRev** parameter for **card_1** has the value **3.141** An **if** step in a test case, for example, might use the **firmwareRev** value to decide which syntax to use for a command.

For details, see "Defining a parameter" on page 735.

2 ***Optional: Organize parameters into a structure***

You can group parameters into structures. In the tree that represents the structure, ***nodes*** are containers with names. Nodes can contain parameters and/or other nodes. The example parameter tree has a node named **slot**. The **slot** node includes two subnodes: **card_1** and **card_2**. The **card_1** node holds parameter definitions for the **cardType** and **firmwareRev** parameters.

For details, see "Creating structure for parameters (working with nodes)" on page 739.

3 ***Optional: Specify custom merge settings***

Merge settings specify, for example, which value to use for a particular parameter when the test case and its local testbed include a parameter with the same name but with different values. The set of parameters and values that result after merging is the set that is used for execution.

Note You can specify a Global parameter file. The parameters that you define here on the **Parameters** page are merged with the definitions in the Global file, as described in "How parameter definitions from multiple sources are merged at run time" on page 752.

Define Custom Types

In the Test Case editor, Testbed editor, or Session Profile editor, the **Custom type** tab allows you to define custom parameter type with a set of named value elements. The custom parameter type and elements you define displays as an option that you can select to indicate the parameter types and values. See "Custom Types" on page 750.

Optional: Save the parameter definitions as a parameter file

In the Test Case editor, Testbed editor, or Session Profile editor, click Save Parameters As... to save the current set of parameter definitions as a parameter file. (While working on a parameter file in the Parameter editor, click **Save** . Parameter files are described in "Parameter files" on page 746.)

Defining a parameter

1 Click the **Parameters** tab on the appropriate editor: Test Case editor, Testbed editor, Session Profile editor, or Parameter editor.

2 Uncheck **Show the parameter values that will be used for execution.** (We discuss this setting in Step 6.)

3 One of the following:

- To add the parameter as a child of an existing node, select the node and click **Add Child** . (Nodes are containers for parameter definitions and are discussed in "Creating structure for parameters (working with nodes)" on page 739.)
- To add a parameter at the same level as an existing parameter (a sibling), select the sibling parameter and click **Add** . The parameter is added after the sibling in the same node.

4 Double-click in the **Name** cell to specify the **Name.** Click in the **Value** and **Description** cells to specify settings as described here:

Name	Specify a friendly name for the parameter (for example, **port**, **slot**, **deviceID**, or **IPaddress**). The name appears in the Data view when you execute a test case. In test case steps, to refer to a parameter that is in a container, use **container_name/parameter_name** syntax. For example, use the following syntax in a **param** command field replacement of the **firmwareRev** parameter in the **Card_1** node: **[param Card_1/firmwareRev]**
Type	Select a parameter type from the dropdown list. Options: Text, Boolean, Integer, Double or Custom type Default: Text An option that displays in addition to **Text, Boolean, Integer,** or **Double** is a custom type parameter defined via the **Custom types** tab (see "Custom Types" on page 750). **Note** Test cases created prior to iTest Release 6.0 provides only the Text type field.

Value	Specify the value of the parameter. (Nodes cannot have a **Value**.) Each of the parameter type value are validated to ensure that you have entered an appropriate value allowed for the **Type** selected. For example, **Boolean** type accepts a **True/False** value. • If you have selected a custom **Type**, the **Value** dropdown list shows the values you entered for the custom type. (see "Custom Types" on page 750). • You may also enter an undefined value for the type. However, an error icon and message (tool tip) displays when you enter an invalid value. • Once you specify a value, it also appears in the Data view, where you can modify it when execution is paused or loaded for execution. You can create an empty parameter (that is, a parameter with no value). Multiple values for a single parameter are useful, for example, in **foreach** loops. To create a parameter with multiple values, do either of the following: • Create multiple parameters with the same name. • For a single variable name, type each value and separate the values using spaces.. If any single value contains a space, then use double quotes around the value.
	Dynamic test case parameter value: You may want to use a query language for defining a test case parameter, which is resolved to a resource property (concrete value) when the test case is run. **For example:** Name=resParameter and Value=$(topology/resources/PC/inventoryName) is resolved to the resource property when running a test case. See details of using the **Property Query Language** below.

Property Query Language

Dynamic property value is a placeholder, which is substituted with the value of a resolved property query:

```
${property_query}
```

Structure of a query:

```
<LOCATION>/resources/<RESOURCE_NAME>/[PORT_NAME/]<PROPERTY_NAME>
```

Where parameters are following:

- **LOCATION** - a data origin, available options:
  - inventory;
  - topology.
- **RESOURCE_NAME** - a name of a resource;
- **PORT_NAME** - optional, a name of resource port;
- **PROPERTY_NAME** - a name of a property.
- **Escape character '\'**:

  Use the backslash character to escape a single character or symbol. Only the character immediately following the backslash is escaped.
  - To use the forward slash (/), use the backslash (\)t o escape. For example, "**A/B"** is escaped to **"A\/B"**.
  - To use a backslash (\) character, use a backslash (\) to escape. For example, "**A/B"** is escaped to **"A//B"**.

Examples:

```
${inventory/resources/Server #1/ipAddress}
${inventory/resources/Cisco Switch #1/Fa0\/1/Port Speed}
${topology/resources/My Virtual Machine/OS Family}
```

Resolving of Property Query

- Depending on the LOCATION, a search for resources, ports and properties is performed in different places:
  - **inventory**: search is performed against Inventory;
  - **topology**: search is performed against a topology TBML file which is associated with a reservation.

  If a property is not found by a query then a test case execution is failed.
- Resolution restrictions for inventory's resource:
  - Password properties are not resolved for a security reason.
  - Attachment properties are not resolved as well.

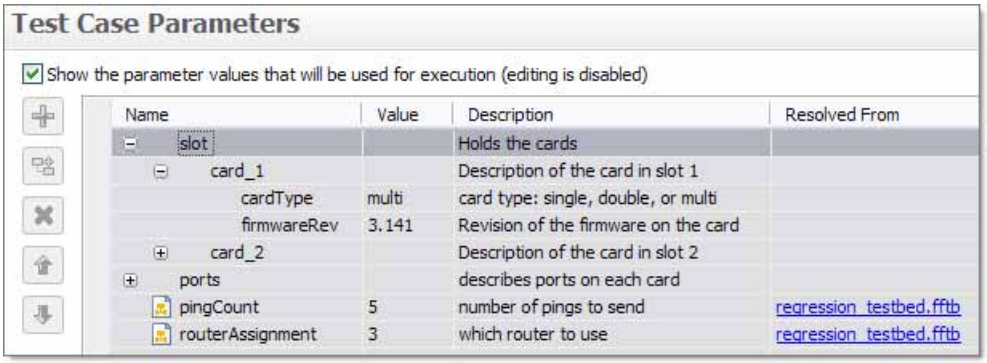
Description	Optional. Describe the function and use of the parameter. The text also appears in the Data view to help you when setting parameter values while executing, pausing, or single-stepping the test case.

5 Optional. **Mask the value.** Check the box for sensitive information (for example, a password) that should be hidden from view in any user-visible windows. See "Masking a parameter's value" on page 744.

6 Optional. **Specify Advance Merge settings.** Merge settings specify, for example, which value to use for a particular parameter when the test case and its local testbed include a parameter with the same name but with different values. The set of parameters and values that result after merging is the set that is used for execution. Typically, you do not need to

change the default settings. See "How parameter definitions from multiple sources are merged at run time" on page 752.

To view the parameter and values that iTest will use at runtime, check **Show the parameter values that will be used for execution**.



The page view switches to a read-only table of parameters and values. For details, see "Previewing the runtime parameter settings while you develop a test case" on page 753.

7 Optional. **Save the parameter definitions as a parameter file**. One powerful way to cause iTest to use a particular set of parameter definitions when executing any test case is to create a parameter file. A parameter file is a collection of parameter definitions and (optionally) references to additional parameter files. See "Creating a parameter file" on page 746.

Tools on the Parameters page

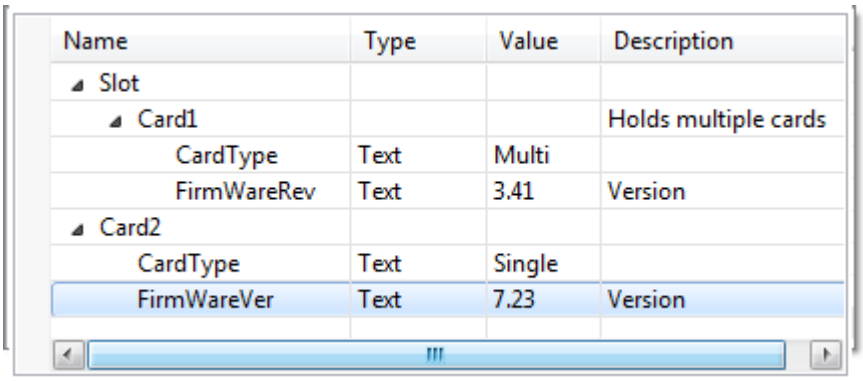
	**Add**	Add a new parameter at the same level as the selected node or parameter.
	**Add Child**	Add a child to the selected node. Nodes are not parameters; nodes contain parameters. If you select a parameter and click **Add Child**, then the parameter becomes a node and cannot have a **Value**.
	**Remove**	Delete the selected parameter. When you delete a node, its children are also deleted.
	**Move Up / Move Down**	Parameters: Move the selected parameter one up or down in the list. Nodes: Move the selected node one up or down in the list. Nodes remain at their current indentation level. Nodes move above or below the adjacent node, not into the adjacent node.
	**Save Parameters As**	Save the parameter definitions into a parameter file. You can use the Parameter editor to edit a parameter file to include additional parameters and, optionally, other parameter files. In addition, you can specify that a particular parameter file is the Global parameter file — to be used by any test case. See "Parameter files" on page 746 and "Global parameter file" on page 751.

Creating structure for parameters (working with nodes)

Nodes are containers for parameter definitions. Nodes are not parameters; nodes contain parameters (and, optionally, other nodes). You will find that using structure helps you to make powerful use of parameter values in your tests.

Note Because nodes are not parameters, they cannot have values.

Example



- The example parameter tree has a node named **slot**. The **slot** node includes two subnodes: **card_1** and **card_2**

- The **card_1** and **card_2** nodes each hold two parameters with identical names: **cardType** and **firmwareRev**
- The tree also includes a top-level node named **ports**

To add a node

1. Click the **Parameters** tab on the appropriate editor: Test Case editor, Session Profile editor, Testbed editor, or Parameter editor.
2. Select the node that the new node should be a child of. For a top-level node, select the appropriate location in the tree.
3. Click  to add a node as a child of the selected node. (The node is first added as a parameter, but as soon as you complete the next step, it becomes a node.)
4. Select the newly-added item and then click  . The parameter is converted into a node (the parent of the new node). The **Value** setting is dimmed because nodes do not have values.

Syntax for referring to a parameter in a node

In a test case step, to refer to a parameter that is in a node, use **node_name/parameter_name** syntax.

In the example, there are clearly two distinct **firmwareRev** parameters. To distinguish between them in test case steps, you refer to them as **slot/card_1/firmwareRev** and **slot/card_2/firmwareRev**

For example, use the following syntax in a **param** command field replacement of the **firmwareRev** parameter in the **card_1** node: **[param slot/card_1/firmwareRev]**

Quick Facts: Where you can define and use parameters

You can define parameters and custom types in the following places:

Test case file (.fftc)

Scope: Visible only within the same test case.

Testbed file (.fftb)

Scope: Visible to all test cases that use the testbed

- Testbed files can be referenced in **run** steps
- Testbed files can be used in test suite files (.ffts)
- A testbed file can be passed as an argument to itestcli

Parameters file (.ffpt)

Scope: Visible globally when a parameters file is marked as **global**

- Parameters files can be referenced in **run** steps
- Parameters files can be used in test suite files (.ffts)
- A Parameters file can be passed as an argument to itestcli

Session profile (.ffsp)

Scope: Visible only to the session steps that directly call the session profile or reference/inherit the session profile

Using Parameters in Properties or Steps

Inserting a parameter into a property or test case step

This topic describes the use of the **Insert Parameter** dialog box to insert a **param** or **profile** command into a test case step or property (that is, any field that supports field replacements).

- You can insert a **param** command or a **profile** command into the field.
- You can create a new parameter in the test case and then insert a **param** command that uses the new parameter (but you cannot set advanced properties for the new parameter).
- You cannot edit existing parameters using the **Insert Parameter** dialog box. Instead, use the **Parameters** page to set the value and advanced properties of a parameter. Parameters can be defined in the test case, in the testbed, in another test case that loaded as a result of a foreign procedure, or in the session profile associated with the step. For instructions on defining parameters, see "Working with parameters: The Parameters page" on page 733.

About the 'param' and 'profile' commands

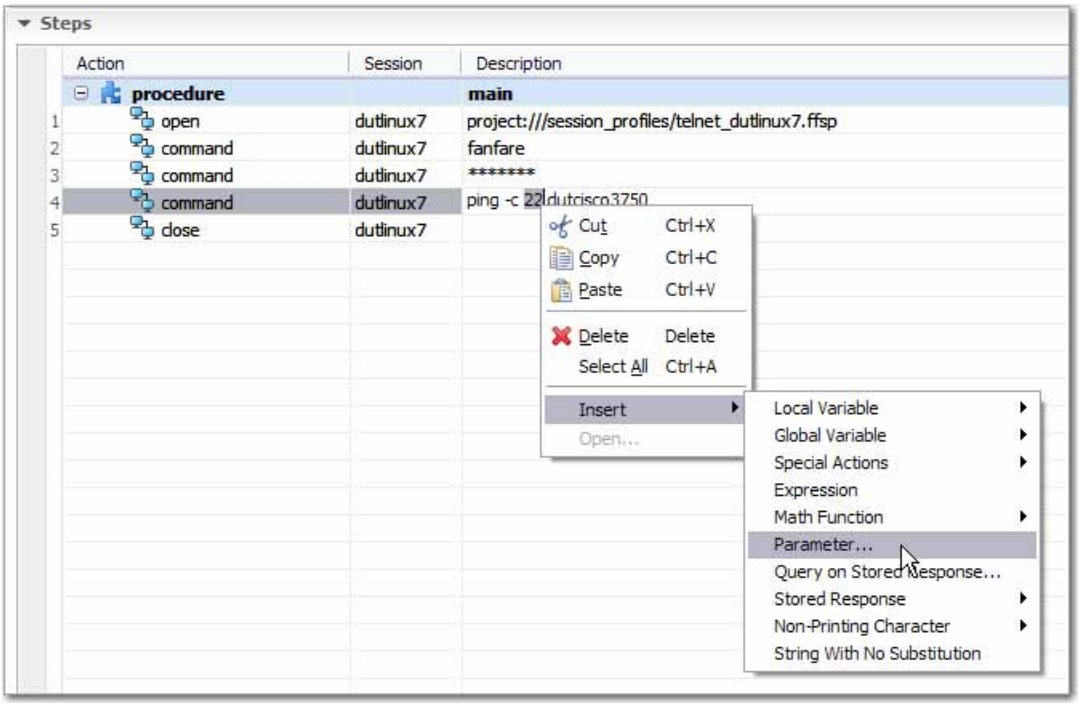
- The **param** command returns the value of a parameter defined in a test case, testbed, or parameter file.
- The **profile** command returns the value of a parameter defined in a session profile.

In this example, **[param ping_count]** is replaced at runtime by the value of the **ping_count** parameter.

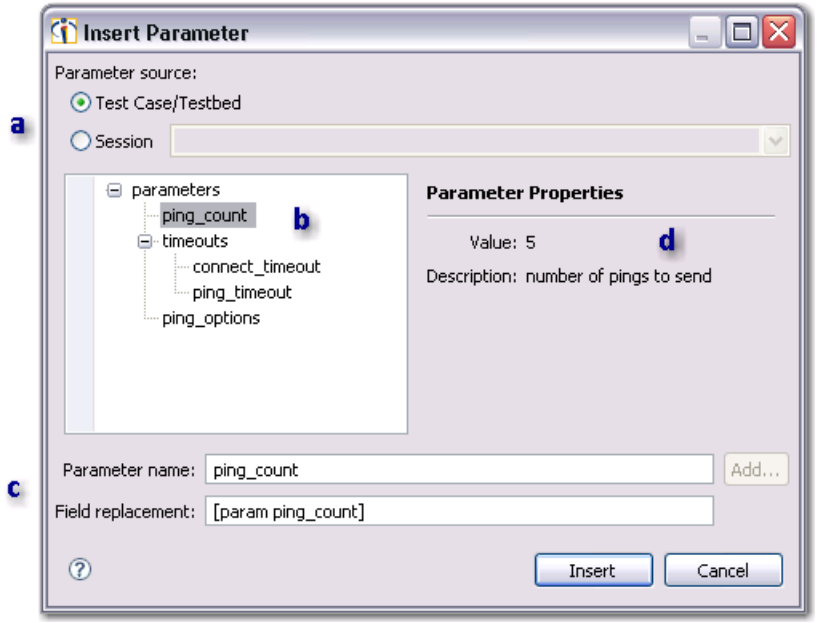
To insert a 'param' or 'profile' command

You can either insert the command at the cursor location or replace selected text. Follow this procedure:

1 In the test case cell or property field, select the text or click in the field. In this example, we'll replace the hard-coded ping count of **22** with a parameter that specifies the ping count at runtime. We select the number 22, right-click, and then select **Insert > Parameter**.



2 The **Insert Parameter** dialog box opens.



a First, specify whether to insert a parameter that is defined in the test case or testbed or to use a parameter that is defined in the session profile associated with the current step's session.

Select either **Test Case/Testbed** or **Session**. If you specify **Session**, then select the session profile from the list.

b This tree view displays the structured list of parameters defined in the specified test case/testbed or session.

Advanced users: If you configured **Advanced merging** settings for how to merge test case and testbed parameters, then the tree reflects the result.

Select the parameter.

c When you select a parameter from the list, the parameter's name and the field replacement for the **param** or **profile** command appear here.

This is the field replacement that will be inserted.

d To help you select the correct parameter, the **Parameter Properties** section displays the value and description of the selected parameter.

Tip Once the parameter appears in the list, you can double-click it to insert the field replacement.

3 The **param** command field replacement now appears in place of the number 22, as shown here. At runtime, the **param** command will be replaced with the value of the **ping_count** parameter.

Defining a new test case parameter and inserting it

Note Using this method, you can add parameters to test cases only — not to testbeds or to session profiles.

1 In the **Insert Parameter** dialog box, select **Test Case/Testbed**.

2 Type the name of the new parameter into the **Parameter name** text box.

3 Click **Add**. Follow the directions for the **Add Parameter to Test Case** dialog box.

Advanced users: Inserting a session profile parameter when the session profile is resolved at runtime

In some test case designs, session profile information for an **open** step is held in a parameter so that the identity of the session profile is determined at runtime. Later in the test case, a step might use a parameter that is defined in the — at design-time, unknown — session profile.

Follow this procedure to insert a parameter that is defined in the session profile—the session profile that will not be resolved until runtime.

Note You will have to know the name of the parameter before you start.

1. Create an **open** step in the test case that uses a **param** command in the **Description** cell to specify the session profile.
2. Later in the test case, for the step that will use the parameter, select the text or click in the field.
3. In the **Insert Parameter** dialog box, select **Session**.
4. In the **Session** text box, type the **param** command that appears in the **Description** cell for the **open** step. Notice that, because you cannot specify a particular session profile, no parameters appear in the list box. For this reason, you have to know the name of the parameter in the next step.
5. Type the name of the parameter into the **Parameter name** text box.
6. Click **Insert**. An appropriate **param** command is inserted.

Masking a parameter's value

While defining or editing a parameter, you can mask its value — specify that the value should be hidden from view in any user-visible windows (for example, a password). iTest performs the following actions for masked parameter values to ensure that the value remains confidential:

- Encrypt the parameter value
- Display the value as asterisks (********) in any editor, view, or report
- The **param** and **profile** commands never decrypt any parameter whose value is masked. As a result, the value never appears in clear (unencrypted) form in any file or in any editor, view, or report visible to a user

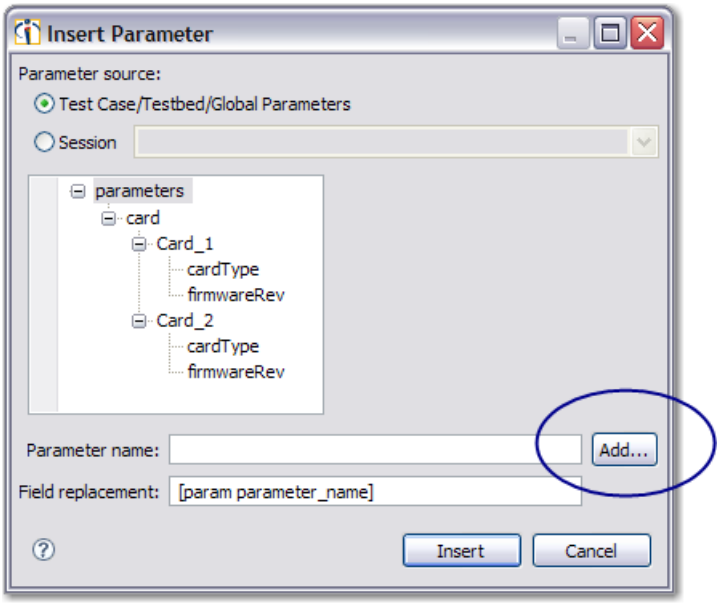
Important To protect the values of masked parameters: If a user attempts to unmask a parameter by selecting it and then unchecking the **Mask the value** check box, the value of the parameter is deleted.

To mask a value

1. On the **Parameters** page of the appropriate editor, check **Mask the value**.
2. When you create a command string with a field replacement such as **[param my_password]**, the iTest interpreter returns the encrypted string. To notify the executing test case to use the decrypted string, you must specify the **Command uses encrypted parameters** property for the step.

Adding a parameter definition while inserting parameters

While inserting a parameter into a test case step, you may want to create a new parameter. This dialog box appears when you click **Add** on the Insert Parameter dialog box.



When you click **OK**, the new parameter is added at the end of the list of test case parameters.

Note The parameter is not actually created until you click **Insert** on the **Insert Parameters** dialog box.

You can specify only the following properties on this page. To specify advanced properties like inheritance rules, use the Parameters page.

Name	Specify a friendly name for the parameter (for example, **port**, **slot**, **deviceID**, **IPaddress**). The name appears in the Data view.
Value	Specify the value of the parameter. The value appears in the Data view, where you can modify it when execution is paused or loaded for execution. • You can create an empty parameter (that is, no value). • To create a parameter with multiple values, create multiple parameters with the same name. Multiple values for a single parameter are useful, for example, in **foreach** constructs.

Description	Optional. Describe the function and use of the parameter. The text appears in the Data view to help you when setting parameter values while executing or single-stepping the test case.
Mask the value	Check the box for sensitive information that should be hidden from view in any user-visible windows (for example, you might mask passwords). iTest performs the following actions for masked parameter values to ensure that the value remains confidential: • Encrypt the parameter value • Display the value only as asterisks (********) in any editor, view, or report • The **param** and **profile** commands never decrypt any parameter whose value is masked. As a result, the value never appears in clear (unencrypted) form in any file or in any editor, view, or report visible to a user

Parameter Files: Centralizing parameter definitions

Parameter files

You can define parameters and their values in several places in iTest. One powerful way to cause iTest to use a particular set of parameter definitions when executing any test case is to create a ***parameter file***.

Parameter files are often used to parameterize the overall operation of a test case. A parameter file is a collection of parameter definitions. In addition, a parameter file can *include* one or more other parameter files (the parameter definitions in the included files are added to the local definitions). Parameter files contain XML with virtually any schema and use the **.ffpt** filename extension.

Global parameter file

You can specify that a particular parameter file is the Global parameter file. The only parameter file that is automatically used by test cases is the Global parameter file. The Global parameter file applies to all test cases in a workspace. See "Global parameter file" on page 751.

Important There is no way to specify a particular parameter file for a test case. You can, however, use iTestRT or itestcli to specify a particular parameter file (that is, not the Global parameter file) for use by a particular test case.

Creating a parameter file

1 You can save a set of parameter definitions as a parameter file from either of the following starting points:

- **Saving existing parameter definitions as a parameter file**: While working on the **Parameters** page in the Test Case editor, Testbed editor, or Session Profile editor, click

. The properties on the **Save Parameters As** page are described in "Creating a parameter file" on page 746.

- **Creating a new parameter file:** In iTest, click **File > New > Parameter File**. On the **New Parameters** page, specify the path (**Container**) and **File name** for the file. Click **Finish**. The file is created and opened in the Parameter editor.

2 Optional: On the **Parameters** page, define parameters for the parameter file. See "Working with parameters: The Parameters page" on page 733 for details.

3 Optional: On the **General** page, provide a **Headline** and **Description** as described in "Parameter editor: General page" on page 748.

4 Optional: On the **Include** page, attach parameter files whose parameter definitions should be included in this parameter file. See "Parameter editor Include page: Including additional parameter files" on page 748.

5 Click **Save** . iTest saves both the parameter definitions and the references to the included parameter files into the new parameter file.

Saving parameter definitions as a parameter file

While working on the **Parameters** page in the Test Case editor, Testbed editor, or Session Profile editor, you can save the current set of parameter definitions as a parameter file:

1 Click Save Parameters As... .

2 Specify the following settings for the new parameter file. iTest saves the parameter definitions and references to the included parameter files into the new parameter file.

Destination Folder	Specify where to save the file. **Workspace folder:** Save the file in a folder in the current workspace. iTest users that use the workspace will see the folder (and the new test report file) in the iTest Explorer. The default folder is **parameter_files**, but you can specify any folder in the workspace or specify a new folder. If you specify a new folder, iTest creates it and then adds the file to it. **File system folder:** Save the file in the specified folder (typically, outside of the current workspace). If the file is saved outside of the workspace, iTest users that use the current workspace will not see the folder (or the new test report file) in the iTest Explorer. Instead, use the operating system's methods for accessing files.
File name	iTest provides a unique, numbered name for the file (**parameters<n>.ffpt**, where **n** is a number). You can modify the name as needed.
Open file in the Parameter editor after saving	Check the box to display the file in the Parameter editor once it has been saved.

Pages on the Parameter editor

Parameter editor: General page

Use the **General** page to supply text that helps test case developers to understand how the parameter file is used.

Headline	Optional. Type a one-line description that documents the usage and function of the parameter file. The text appears in the Favorites view to help you when selecting a parameter file.
Description	Optional. Type text that describes the parameter file to make its usage clear to coworkers.

Parameter editor: Parameters page

Use the **Parameters** page on the Parameter editor to edit individual parameter definitions in the parameter file.

You can add, delete, and change the order of parameters. For instructions on defining and editing parameters, see "Working with parameters: The Parameters page" on page 733.

Important The **Parameters** page that appears on the Parameter editor differs from the **Parameters** page on the other editors (the Test Case editor, the Testbed editor, the Session Profile editor) in the following way: While you work in the Parameter editor, you are editing a parameter file. While you work in one of the other editors, you are editing individual parameter definitions for the current document (test case, testbed, or session profile).

Parameter editor Include page: Including additional parameter files

On the **Include** page, you specify parameter files whose parameter definitions should be included in this parameter file.

So, at run time, the test case uses both the parameters that you define on the **Parameters** page and the parameters that are defined in the included parameter files (the files in the list). The files are merged in the listed order. For additional details, see "Adding a parameter definition while inserting parameters" on page 745.

Including a parameter file into a parameter file

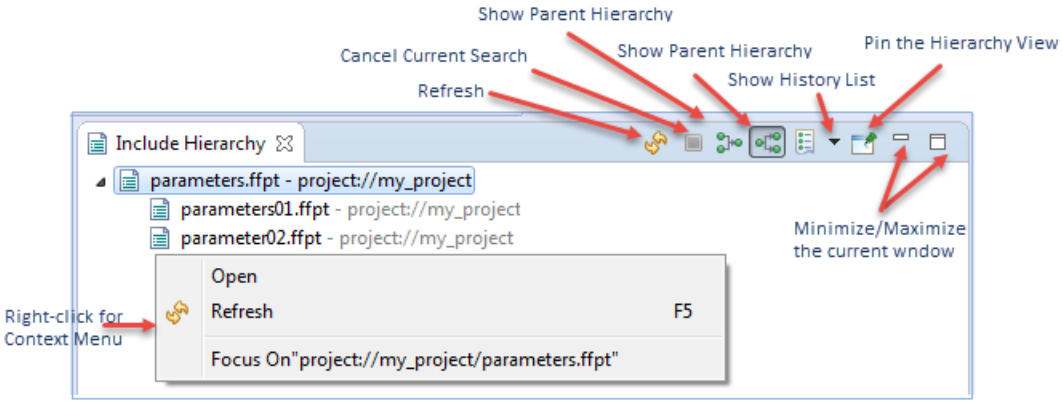
1. Click  to add the URI of a parameter file to the list.

   Any custom parameter types defined in the children Parameter files are available for use from within the parent Parameter file.

2. The order of the files is important because the files are merged in the order in which they appear in the list. Select a URI and click  or  to move it. For a description of how parameter definitions are merged, see "How parameter definitions from multiple sources are merged at run time" on page 752.

3 Click **Show Hierarchy** to display the hierarchy of the included parameter files on the **Include Hierarchy** page.

The **Include Hierarchy** page displays the parent Parameter file and a list of children parameter files that include settings from other files. (That is, files added in Step 1.)



Command	Description
Refresh	Refresh the selected elements and their direct children.
Cancel Current Search	Cancels the current search (useful for long running searches).
Show Parent Hierarchy	Displays all parents of the selected element.
Show Child Hierarchy	Shows all parameter files used by the currently selected parameter file.
Show History List	Displays a history of previously displayed hierarchies.
Pin the Hierarchy View	Pins the current view and allows you to open multiple hierarchy views at the same time.
Righ-Click	Opens a context menu with these options:
	Open: Open selected element in the default editor (Parameter Editor)
	Refresh: Refresh the selected elements and their direct children.
	Focus On: Focus Hierarchy View on the selected element.

4 Other actions:

- To delete a URI from the list, click .
- To edit the selected parameter file, right-click the URI and select **Open Parameter File**.
- To change the URI from a relative URI to another form, right-click the URI and select **Edit File URI**. (The default is relative to the parameter file that you are currently editing.)

Custom Types

In the Test Case editor, Testbed editor, or Session Profile editor, the **Custom type** tab allows you to define custom parameter types and their values See "Quick Facts: Where you can define and use parameters" on page 740.

Defining Custom Types

On the **Custom types** tab, click  to add a new custom parameter type and the name field becomes available. Enter the custom type name and details as described below.

Name	Provide a meaningful, short name. This string appears as an option in the **Type** dropdown list on the Parameter editor. See Step 4 on page 735, section "Working with parameters: The Parameters page".
Details	Click  and the **Details** panel appears. Enter name of the **Type value** and description You may define multiple item names and description.
	Name: Enter a name of the **Type vlaue.** This name will appear as a value when the custom type name (entered above) is selected on the Parameters tab. See Step 4 on page 735, section "Working with parameters: The Parameters page". **Note** You cannot enter duplicate names.
	Description: Enter text to describe the item.

Custom types page toolbar

 **Add**	Add a new named parameter type definition.
 **Remove**	Delete the selected parameter type definition.
Move Up / Move Down	Move the selected parameter type/value definition up or down in the list. When you select a parameter type and then select a value, the dropdown list displays the values in the listed order.

Note The custom parameter type and elements you define displays as an option that you can select to indicate the parameter types and values.

The following applies when working with parameter files that has custom Type and Value defined:

- A parent parameter file has access to the custom parameter types and values defined in the included children parameter files.
- A child parameter file and parent parameter files cannot define identical custom Type name with different details/item name (value).

  In such cases, when you select the custom type (identical Types with different values) in the parent file, a warning message displays saying that "custom type is defined in the dependent files with different items".

- Inheritance of parameters

  A session profile (e.g., Session_one) referencing another session profile ((e.g., Session_two via **This session profile inherits settings from another session profile**) has access to the custom parameter types and values defined in the referenced session (e.g., Session_two) profile.

Global Parameters

Global parameter file

When a particular parameter file is the Global **parameter file**, then any executing test case uses the parameters defined in the file. In addition, if the file references other parameter files, then the parameter definitions in the referenced files are included.

Note The only parameter file that is automatically used by test cases is the Global parameter file. Using iTest, there is no way to specify a particular parameter file for a test case. You can, however, use iTestRT or itestcli to specify a particular parameter file (that is, not the Global parameter file) for use by a particular test case.

The Global parameter file applies to all test cases in a workspace.

How parameter definitions from multiple sources are merged at run time

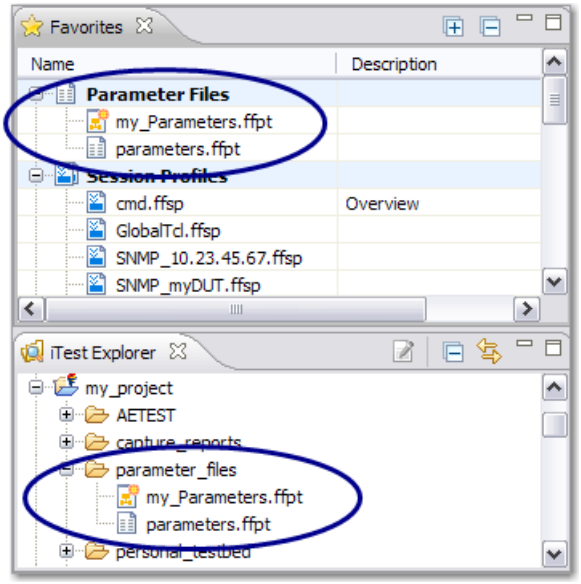
Parameters can be defined in several places in addition to the Global parameter file. iTest combines the parameter definitions from all sources before execution. See "Adding a parameter definition while inserting parameters" on page 745.

Specifying a parameter file as the Global parameter file

In the iTest Explorer or Favorites view, right-click the parameter file and select **Set As Global Parameter File**.

To undo the setting, right-click the Global parameter file and select **Clear Global Parameter File** or specify that another parameter file is the Global parameter file.

The icon for the Global parameter file is marked with an orange icon to distinguish it from other parameters.



Merging parameter definitions from multiple sources

How parameter definitions from multiple sources are merged at run time

Overview

You can think of the parameter value that you define in the session profile as the default or base value. At runtime, for an example parameter named **param1**, the following changes occur:

1. If you specified a value for **param1** in the testbed, iTest uses the value from the testbed, and then:
2. If you specified a value for **param1** in the test case, iTest uses the value from the test case, and then:
3. Finally, if you specified a value for **param1** in the Global parameter file, iTest uses the value from the Global parameter file

This clearly defined order (in which a parameter value is overridden) is called the merge order.

So, if you have specified a Global parameter file, then (regardless of any values in the other files) the value specified in the Global parameter file is the value that is used at runtime. Because this is true whenever you have specified a Global parameter file, you do not need to specify that the value in the Global parameter file should be used. That is why the Global parameter file does not appear as a source of parameter values in the **Insert Parameter** dialog box.

How parameters are accessed

Based on where they are defined, parameters are accessed as follows (using the default inheritance settings). You also have the option to specify "Advanced merging behavior for parameters" on page 755.

Session Profile editor (also called the New Session page)	When you create a session profile and base it on an existing profile, the new profile inherits the parameter definitions in the existing profile. When you specify a session profile in an **open** step in a test case, any step in the session that was opened with the profile uses the parameters and values defined in the profile.
Test Case editor	Parameters that you define for a test case can be accessed by any step in the current test case. Test case steps can make use of param field replacements (the **[param paramName]** command) to overwrite settings made in testbeds or session profiles.
Testbed editor	Parameters that you define for a testbed can be accessed by any step in a session that is based on a session profile specified in the testbed.

Merge order

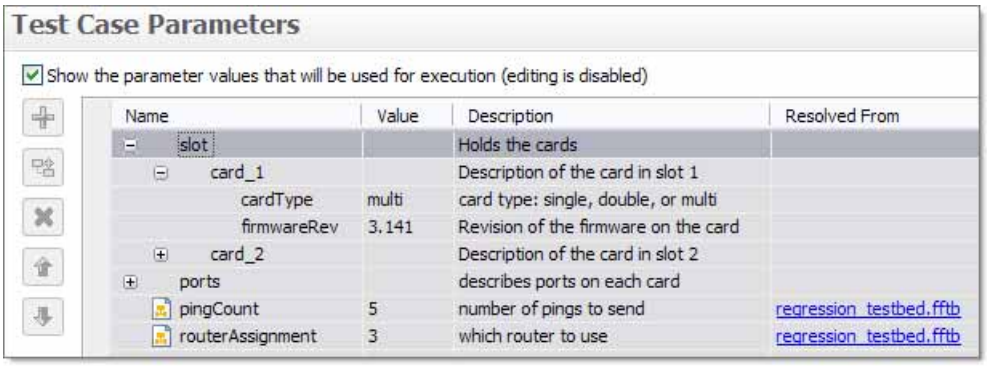
Parameters can be defined in several places. iTest combines the parameter definitions from all sources before execution. Here's how it works with default settings for the **Advanced merging behavior** properties (if no parameters are specified in a particular place, then the step is skipped):

1. First, parameters defined in the Global **parameter file** are loaded onto the heap.
2. Parameter files that are **included** with in the Global file are loaded (again, the parameters defined in each file are loaded first, followed by any referenced files, recursively). The files are loaded in the order in which they are listed in the parameter Global file.
3. The parameters defined in the **test case** are loaded.
4. The parameters defined in the **testbed** to be used for execution are loaded.
5. The parameters defined in the **session profiles** to be used for execution are loaded.

Previewing the runtime parameter settings while you develop a test case

To view the parameters and values that iTest will use at runtime (that is, the parameter set after merging), check the **Show the parameter values that will be used for execution** check box. The **Parameters** page switches to a read-only table of merged parameters and values.

Example



- The view opens with all node structures collapsed. Click ⊞ and ⊟ as needed to view parameters of interest. In the example, we expanded the **slot** and **card_1** nodes. The **card_2** and **ports** nodes are still collapsed.
- The **Resolved From** column displays the source of each parameter (which file it is merged from):
  - Because **cardType** and **firmwareRev** are local parameters (defined in the document that is being edited—a test case in this example), there are no entries in the **Resolved From** cells for them.
  - The **pingCount** and **routerAssignment** parameters are defined in the testbed named **regression_testbed** (it is the local testbed for the test case that we are editing). iTest displays the icon for a testbed document to emphasize that the values are resolved from another document. To view or edit a parameter definition, click the link. The appropriate editor opens to the **Parameters** page.

Restrictions

As described in the following section, not all merged parameters are displayed on the **Parameters** page. To view the full set of merged parameters, load the test case for execution and open the Data view.

Test Case editor

In the Test Case editor, the **Parameters** page displays merged parameters only from the local testbed (including its chain of inherited testbeds). The **Parameters** page does not display the following:

- Parameters that are merged as the result of a **call** step
- Parameters that are merged from the Global testbed file
- Parameters that are merged from the Global parameters file
- Session profile parameters defined against a device within the local testbed

Session Profile and Testbed editors

The **Parameters** page displays merged parameters only from the current file (and its chain of inherited files). The **Parameters** page does not display the following:

- Parameters that are merged from the Global testbed file
- Parameters that are merged from the Global parameters file

Advanced merging behavior for parameters

To determine parameter values to use at runtime, a test case can inherit nodes and parameters from a testbed (either its Local testbed or the Global testbed, as appropriate), a session profile, or a parameter file.

A child test case (a test case that is executed by the **run** command in a parent test case) can inherit nodes and parameters from a testbed and from the parent test case.

By default, parameters with different names are inherited. The advanced mode settings handle special situations where the parameter names in the testbed and the test case are the same.

Default merging behavior

For inheritance purposes, the testbed is the parent of the test case. By default, test cases inherit parameter settings from testbeds. By default, parameters with different names are inherited.

For inheritance purposes, test cases that run child test cases are parents. By default, child test cases inherit parameter settings from parent test cases. By default, parameters with different names are inherited.

How merging works

There are two distinct phases to the merging process:

- **Merge Alignment:** Whether to add the structure or not: How to add nodes and parameters when similar structures are specified in the parent.
- **Merge Value Update:** After the alignment process is complete, overwrite the value or not: How to merge the value of any parameter that has the same name as a parameter defined in the child when similar parameters are specified in the parent.

Merge Alignment setting

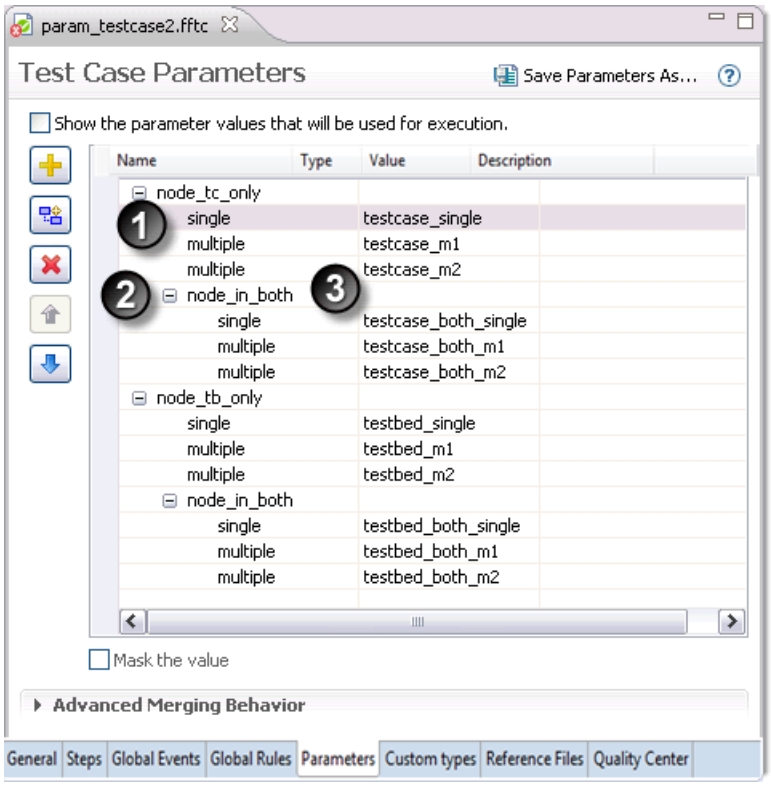
The **Merge Behavior** table that appears below this table presents a summary of how nodes and parameters are merged based on the **Merge Alignment** setting.

Note This setting specifies how to merge parameter definitions. Use the **Merge Value Update** property setting to specify how to merge the values of the parameters.

Inherit choice from parent; if no parent, add if missing/multiple	Default setting. Inherit the **Merge Alignment** setting for the parent node/parameter. If the setting in the parent is the default setting, then use the **Use if present; add if missing/multiple** merge behavior.
Always add	Add any node or parameter that appears in the parent and not in the child.
Use if present; add if missing/multiple	Add any node or parameter that appears in the parent and not in the test child. Add any node or parameter that appears in the parent and also in the test child. Use node if present; add if the node does not exist in the child or if multiple nodes are being merged
Use if present but do not add if missing	Use the node if present in the child, but do not add if missing from the child

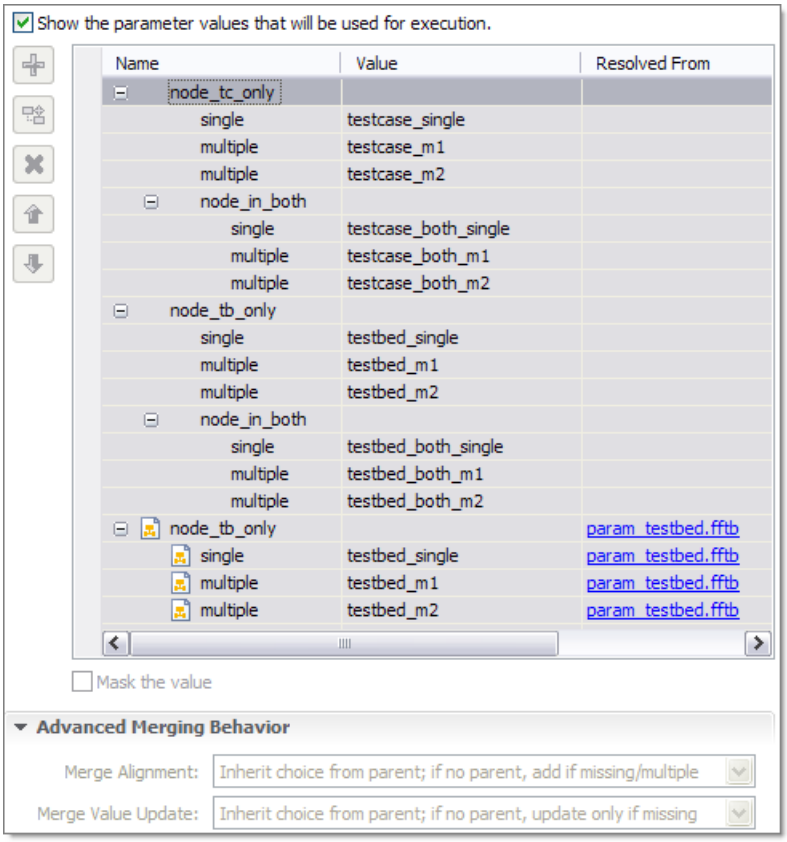
Example

This example test case and its local testbed illustrate how each merging option works.



1. This node appears only in the test case.
2. This node appears both in the testbed and in the test case.
3. As shown in the table, all values in the test case use the text "**testcase**".

- **Results for each Merge Alignment setting (as set in the Advanced Merging Behavior section**



Merge Value Update setting

The **Value Overwrite Behavior** table that appears below this table presents summary of how values are set based on the **Merge Value Update** setting. For the definition of ***node***, see "Creating structure for parameters (working with nodes)" on page 739.

Note This setting specifies how to merge parameter values. Use the **Merge Alignment** property setting to specify how to merge parameter definitions.

Inherit choice from parent; if no parent, update only if missing	Default setting. Inherit the **Value Update** setting for the parent node/parameter. If the setting in the parent is the default setting, then use the **Update value only if missing** behavior.
Do not update value	The value specified for the **Value** property on this page takes precedence. Do not overwrite the value.
Update value only if missing	If the current document does not define a parameter value that is defined in the merging document, then overwrite the parameter values.
Update value only if present	If the merging document includes a parameter with the same name, then overwrite the values.
Always update value	If the merging document includes a parameter with the same name, then overwrite the values.

Value Overwrite Behavior based on the Merge Value Update setting

Here's how values are merged:

	No existing value	Existing value
Do not update value	Do not overwrite	Do not overwrite
Update value only if missing	Overwrite	Do not overwrite
Update value only if present	Do not overwrite	Overwrite
Always update value	Overwrite	Overwrite

CHAPTER 34

Session Builder

Session Builder Overview

iTest session builder enables developers to create custom sessions which can be delivered without concern for their IP being compromised and optionally, to sell and generate income.

This chapter describes using Session Builder to export an existing QuickCall libraries as a new session type, extend the existing session by adding custom command actions, add a license to the new session type (to protect your IP), and share the custom integrations for reuse.

Note You may choose not to extend the existing session and/or re-use the license from the base session.

See Chapter 4, "Session Profiles" on page 71 and Chapter 9, "QuickCalls: Defining and using a library of custom actions" on page 205 for details about creating iTest Sessions and defining QuickCall libraries.

Note Existing sessions and QuickCalls may be used to create custom session types. For example, two custom sessions available, **ADB** (Android Debug Bridge) and **OpenStack Neutron**, are built using the Session Builder.

Creating a custom session type

This topic describes the steps involved in creating a new session type with Session Builder.

- Deriving new session type from native iTest session type and a QuickCall library using REST, Tcl, CMD and other session types. In Addition, indicate initialization procedures for console based-sessions (REST, Telnet, SSH, etc.).
- Generating Session type
- Adding a license to the new session type or Re-using the license from the base session

Building a new Session type

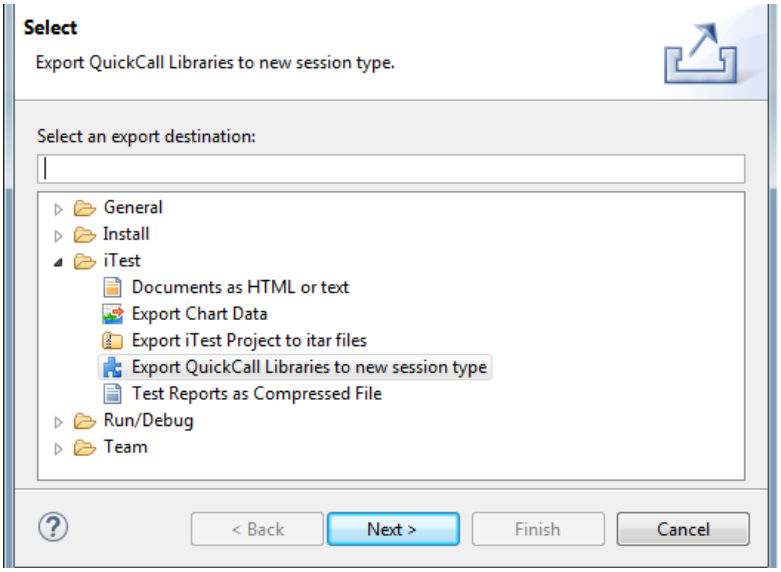
Note Make sure that you have QuickCall libraries defined ("Defining a QuickCall" on page 207) using the required session types, for example, using REST, Tcl, CMD and so on.

Follow these steps to build a custom session type based on the QuickCall library.

Step 1 Export QuickCall Library

These steps describes deriving new session type from native iTest session type and a QuickCall library using REST, Tcl, CMD and other session types.

- Select **File > Export** from iTest GUI, and the **Select** window opens.
- Select option **ExportQuickCall Libraries to new session type** and click **Next.**



- Click **Next** and the **Select QuickCall Libraries** window opens

Step 2 Select QuickCall Libraries and location to save

When the **Select QuickCall Libraries** window opens, select the quick call libraries and specify the location to store the custom session package.

- Select the required QuickCall library from its location (All Projects, a specific project, my_project, or resource).
- Select the QuickCall library(s) and indicate the export location (**Export To:**) by selecting **iTest resource** or **Export to directory**.
  - Selecting **iTest Resource (default):**

    This is the default location. The custom session will be automatically installed and requires restarting iTest for the installation to take effect.
  - Selecting location as **Export to directory:**

    Browse and select the directory to copy the custom session type. This option does not require you to restart iTest as the custom session is not installed automatically. You are

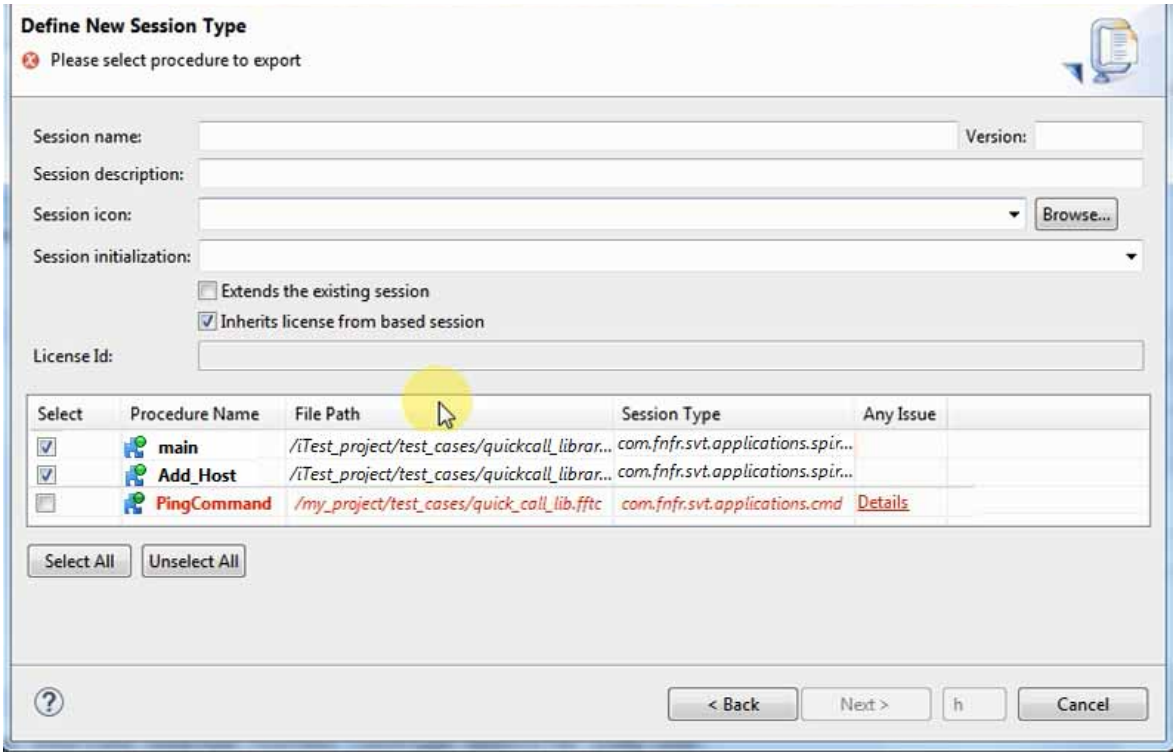
required to User need to install this session Browse to the manually for use. See "Manually Installing Custom Session Type within iTest" on page 781 for details.

- Click **Next** and the **Define New Session Type** wizard displays.

Step 3 Define New Session Type wizard—QuickCall definitions validation before export

When you click **Export QuickCall Libraries to new session type** to define custom sessions, iTest validates the QuickCall libraries for any mismatched datatype defined as an argument description and displays errors on the **Define New Session Type** window. iTest validates exported QuickCall libraries as per the criteria shown in "QuickCall definitions validation criteria" on page 764.

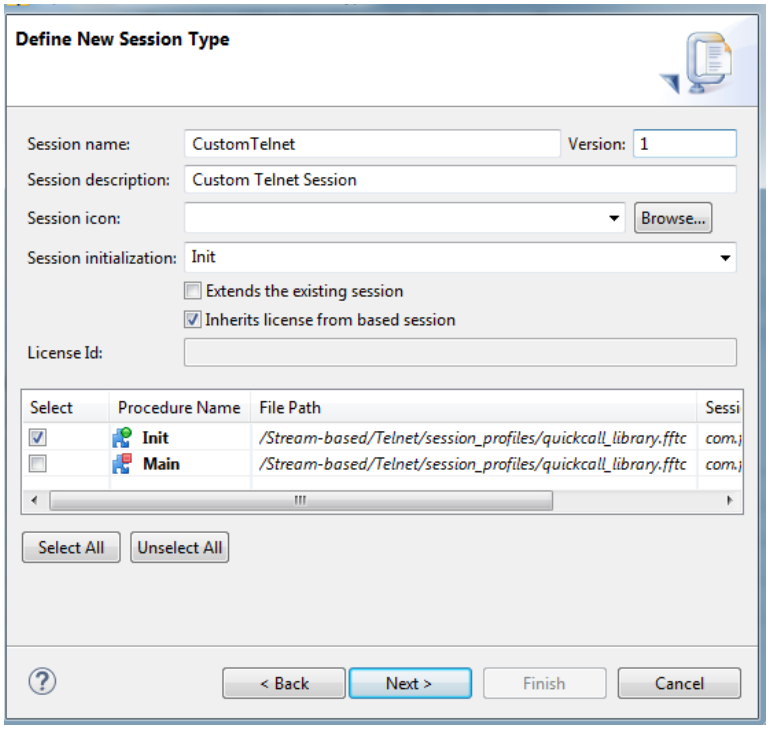
If validation fails, and the **Procedure Name**, **File Path**, **Session Type**, and an **error message** appears in red text, as shown below



Click the **Details** link under the **Any Issue** column to understand the error.

Step 4 Define New Session Type wizard—Complete information and export session

Enter the following details when the **Define New Session Type** wizard displays:



- **Session name**: Enter a meaningful recognizable session type name.
- **Version**: Valid version number is: \d+(\.\d+)*, for example: 0.9, 0.9.1, 1.0, 1.0.2, 2.2, and so on.
- **Session Description**: Enter a description as required.
- **Session icon**: Browse to the location of the icon and select the icon to associate with the session type.
- **Session Initialization**: Displays selection options based on the procedures inluded in the QuickCall session and the option **None**. For example, **Init**, **Main**, and **None** (for the illustration above). Select **Init** to indicate that the custom session includes an initialization/main QuickCall, which initializes the environment settings, authentication, etc, and ensures that the subsequent QuickCalls (Steps) work as required.

  For example, a Telnet session requires you to interact with terminal console and provide the username/password to login to Telnet server.

  Note A QuickCall library based on iTest sessions (REST, Telnet, SSH, etc) require initialization of environment settings, authentication, etc.

- **Extends the existing session**: Generate the new Session type with these options.
  - Not selected (default). When **Extends the existing session** is not selected, the custom Session Builder hides the native session type, and provides a command line interface and allows using the console.

Note Selecting **Extends the existing session** disables any selection in the **Session Initialization** option.

- Selected. When **Extends the existing session** is selected, the Session Builder extends the native session type (using the GUI). For example, use this option to extend REST API with custom commands tailored for a particular need.

Note Both options hide the native libraries.

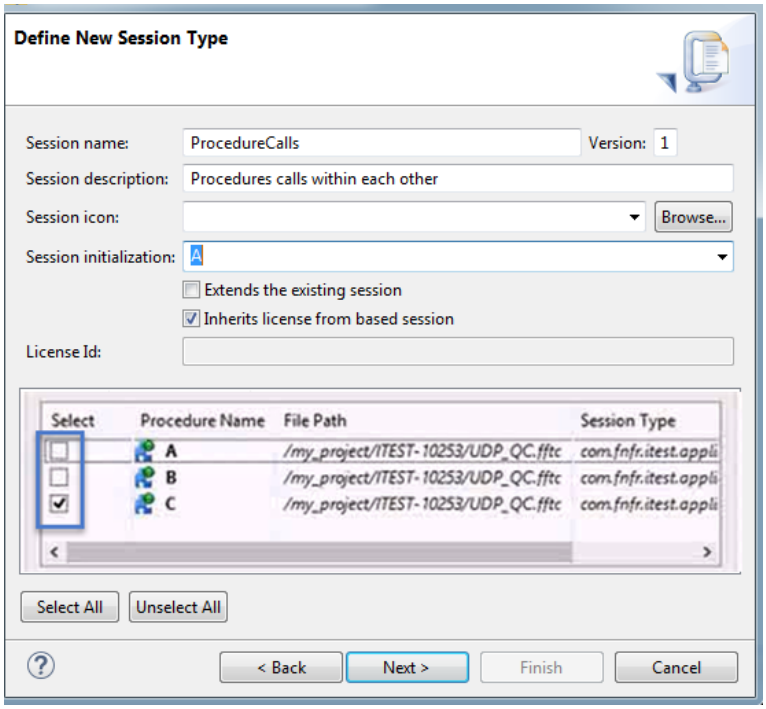
- **Inherits license from based session**: Selected by default and re-uses license from the based session.
- **License ID**: Becomes available only when Inherits license from based session is not selected. This option allows you to add a new license to the new session type.

Tip Contact Spirent customer support to request generating a custom license for your custom session and provide the license key string you wish to use. For example string **iTESTOPENSTACK#33.10.2016** for custom **OpenStack Neutron** session.

- **Select** the required **QuickCall** or **Select All** from the list of usable session type in the Wizard.

  When you also select a single procedure to export, from a list of displayed procedures, all the selected procedure's dependencies will also be exported automatically as session specific commands in the custom session, if any and not selected.

  For example, in a Quickcall with 3 procedures A, B, C, where C calls B, B calls A, and A contains a single step to display the Help command, selecting only procedure C to export, also exports procedures A and B as session specific commands.



- The **Back**, **Next**, **Finish**, and **Cancel** buttons become available only after you complete all the above information. The **Next** button allows you to attach a document to the new custom session.

Step 5 Attach User document to the new custom session

If the **Next** button is available, click **Next** to select the Document for the new session type.

- Attach document as follows:
- Select **Use Online document** and provide a URL of the document location.

OR

- Select **Attach a document**, browse to the document location and select the required text or a PDF document.

Step 6 Complete defining new session type

Click **Finish** to complete defining a new session type. It test validates QuickCall definitions as per the criterea described in "QuickCall definitions validation criteria" on page 764.

If the validation process completes sucessfull, a message displays saying that iTest needs to restart in order to apply the new session type and whether you wish to restart iTest. Click **Yes** to restart iTest.

iTest restarts and the newly installed session type appears in as one of the session types as illustrated in the **Start a New Session** window (on page 769).

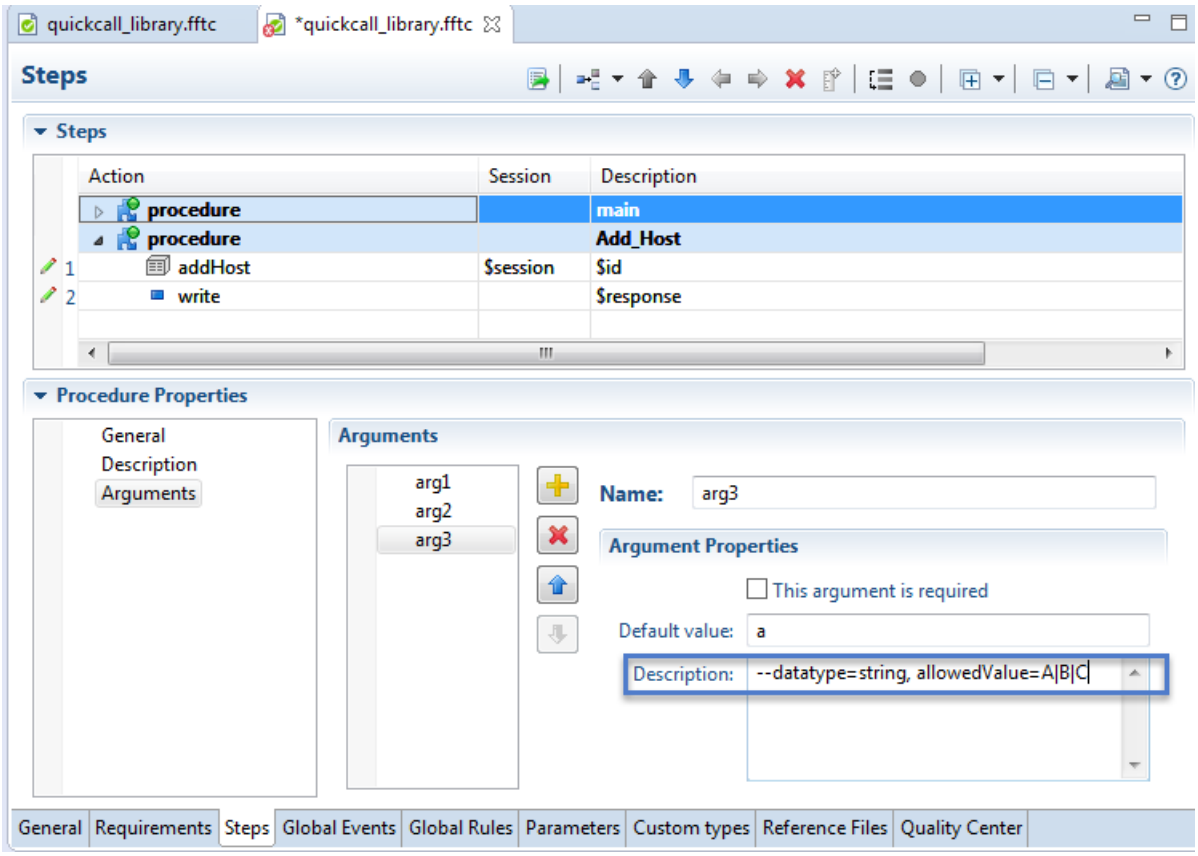
QuickCall definitions validation criteria

iTest Session builder validates custom session commands properties defined in the QuickCall argument description (see "Custom session command properties declaration"). When you export quickcall library, commands for the new session type are validated as per the predefied criteria shown in section "Validation rules applied when exporting Quickcall library" on page 767. After the defined Quickcall argument description passes the validation criteria, they will be developed as commands in the new custom session types as per the rules and properties defined.

The following diagram shows an example of data type defined in argument description:

Note The validation rules and description of custom session commands are separated by this delimitor: **--**.

Argument Description—QuickCall Procedure



Custom session command properties declaration

iTest parses each argument of a Quickcall as a property of the custom session command. The definition in the argument description helps clarify how it may be used when exporting the Quickcall library. By default, the datat ype of each argument is a string and can be defined to contain different data types. For example, --datatype=integer. In addition, the declaration can also provide subsequent validation rules for an argument such as **allowedValue**, **allowedRange**, **allowedLength**, etc. The table below shows how to specify the validation rules for an argument.

Properties	Description	Syntax/example
datatype	Indicates the data type values appropriate for values in this argument. The values must be one of the following: **string, integer, boolean, decimal, anyURI, or dateTime**.	--**datatype**=integer
masked	True or false. True: indicates that the contents of the argument are sensitive (such as a password) and should normally be hidden from view by users. False:indicates that the value may be visible (not masked)	--datatype=string, **masked=true**

Properties	Description	Syntax/example
isMultiline	True or false. Applicable Only if the datatype is string. True: indicates that the string is allowed to span multiple lines. False: indicates that the string is not allowed to span muliple lines.	--datatype=string, **isMultiline=true**
allowedValue	In cases where only a certain fixed set of values are appropriate for the datatype, then the **allowedValues** element will appear once in the parameter declaration for each of these values.	--datatype=string, **allowedValue=A|B|C**
allowedLength	Aplicable for string argument. Establishes a minimum and/or maximum length for the string.	**Syntax** :allowedLength={min_value; max_value} or allowedLength={min_value; } or allowedLength={ ;max_value} **Example**: --datatype=string, **allowedLength={3;5}**
allowedCount	Determines whether the given argument may appear multiple times in the container.	**Syntax**: allowedCount={min_value;max_value} or allowedCount={min_value; } or allowedCount={ ;max_value} **Example**: --datatype=string, **allowedCount={3;5}**
allowedPattern	Indicates that the string arguments are valid only when they conform to a certain pattern.	--datatype=string, **allowedPattern=([\da-fA-F]{2}:){5}[\da-fA-F]{2}**
allowedRange	Valid for numeric arguments (integer and decimal). The **allowedRange** indicates the minimum and/or maximum values that will be considered valid. When both minimum and maximum values are specified and if the minimum value is greater than the maximum value, then the range between these values will be invalid.	**Syntax**: allowedRange={min_value;max_value} or allowedRange={min_value; } or allowedRange={ ;max_value} **Example**: --datatype=integer, **allowedRange={3;5}**
enablementValue	Indicates that the validity of an argument depends on the presence and/or values of other arguments. That is, if the **enablementValue** element is specified in the argument description, then you will be allowed to indicate that this argument is valid only when another argument (by name) within the same container carries a specific value.	--**enablementValue = { parameter: "param1", value : "", enableOn: "equal"}**

Validation rules applied when exporting Quickcall library

Based on the definition in the Quickcall argument description, the export Quickcall wizards will apply validation rules for each Quickcall argument before it can be a custom session command properly. The following rules apply: See "Validation rule for default value" on page 767, "Validation rules for allowedPattern, allowedLength, etc., with datatype" on page 767 and "Validation rules for AllowedRange, AllowedLength, AllowedCount:" on page 768.

Validation rule for default value

In iTest Quickcall, the default value of argument can be specified in the parameter **default value**. When the **default value** is specified and the custom session command property declaration is defined, the validation criteria will be used to validate any conflict. The table below shows the validation criteria and the error message displayed..

Validation criteria	Example	Error Message
The **defaultValue** does not match the **datatype**	dataType=integer, defaultValue=abc	"The value '%s' does not match the data-type %s. "
The **defaultValue** does not match the **allowedValue**	allowedValue=1|2, defaultValue=a	"The defaultValue '%s' is not in the allowedValue list \"%s\". "
The **defaultValue** does not match the **allowedRange**	allowedRange={2;4}, defaultValue=5	"The defaultValue '%s' does not match the allowedRange \"%s\". "
The **defaultValue** does not match the **allowedLength**	allowedLength={2;4}, defaultValue=abcde	"The defaultValue '%s' does not match the allowedLength \"%s\". "
The **defaultValue** does not match the **allowedPattern**	allowedPattern=\\d+, defaultValue=abcde	"The defaultValue '%s' does not match the pattern \"%s\". "

Validation rules for allowedPattern, allowedLength, etc., with datatype

The table below lists the properties and the supported datatype. When you use a property that does not match the datatype, a validation conflict occurs and a validation error message displays. For example, using **AllowedRange** with String (i.e., --datatype=string, allowedRange={3; 5}), displays an error message as follows.

> The datatype 'String' does not support the attribute '**AllowedRange**'. The supported attributes are **AllowedValue**, **AllowedPattern**, **AllowedLength**, **AllowedCount**, **Marked**, **isMultiline** when exporting a Quickcall library.

Properties/DataType	string	integer	decimal	boolean	datetime	anyURI
AllowedValue	x	x	x	x		x
AllowedPattern	x				x	
AllowedRange		x	x			
AllowedLength	x					
AllowedCount	x	x	x	x	x	x
Marked	x	x	x			x
isMultiline	x					
enablementValue	x	x	x	x	x	x

Validation rules for AllowedRange, AllowedLength, AllowedCount:

The properties of **AllowedRange, AllowedLength,** and **AllowedCount** must be defined with the correct systax and structure as follows:

Format/Syntax: {min;max} or {;max} or {min;}

If an inconsistent minimum or maximum values are used, validation criteria/rule displays an error message, as the validation does not meet the defined rules:

Invalid allowed %s: %s. It should be allowed%s={min_value;max_value} or allowed%s={min_value; } or allowed%s={ ;max_value}.

For example, **--datatype=integer, allowedrange={5;3}**, displays an error message as follows:

Invalid allowed range: {5;3}. It should be allowedRange={min_value;max_value} or allowedRange={min_value; } or allowedRange={ ;max_value}.

Using the new custom session type

A new session type developed using the Session Builder is similar to a native iTest session usage. Depending on your selection of option **Extends the existing session** on page 762 (Step 4 “Define New Session Type wizard—Complete information and export session” on page 762), the new session type may be defined for usage as follows:

- Hide the native session type, provides a command line interface and allows using the console.
- Extend the native session type (using the GUI), customize and package with the content library and a new interface.

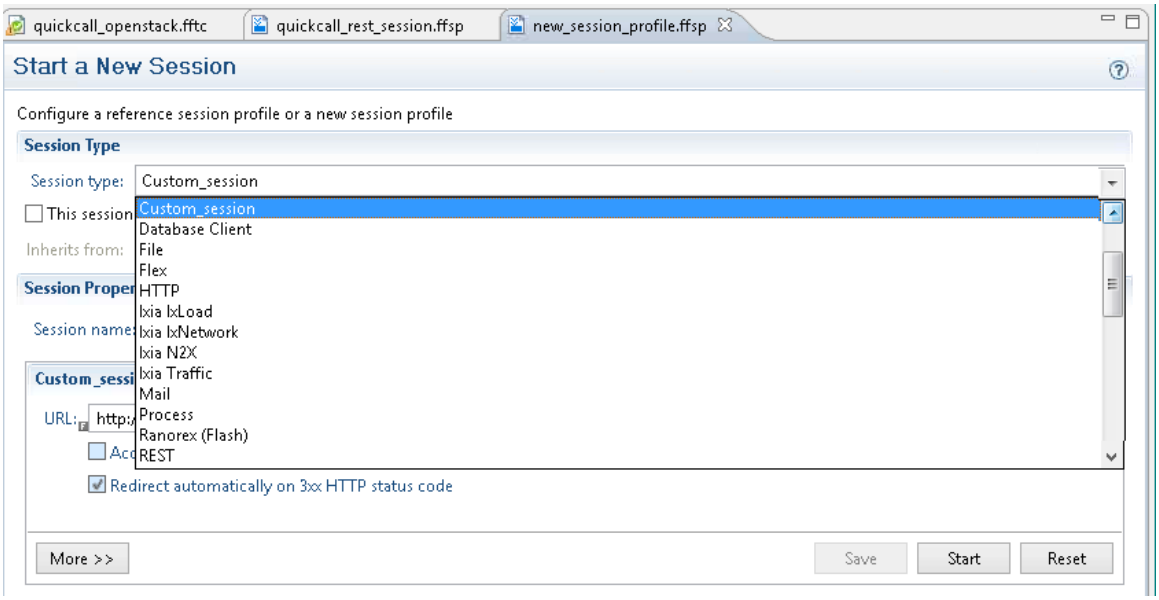
This section includes the following topics:

Using extended custom session

These steps describe opening the new custom session you built (“Building a new Session type” on page 760), executing the session and customizing the session commands.

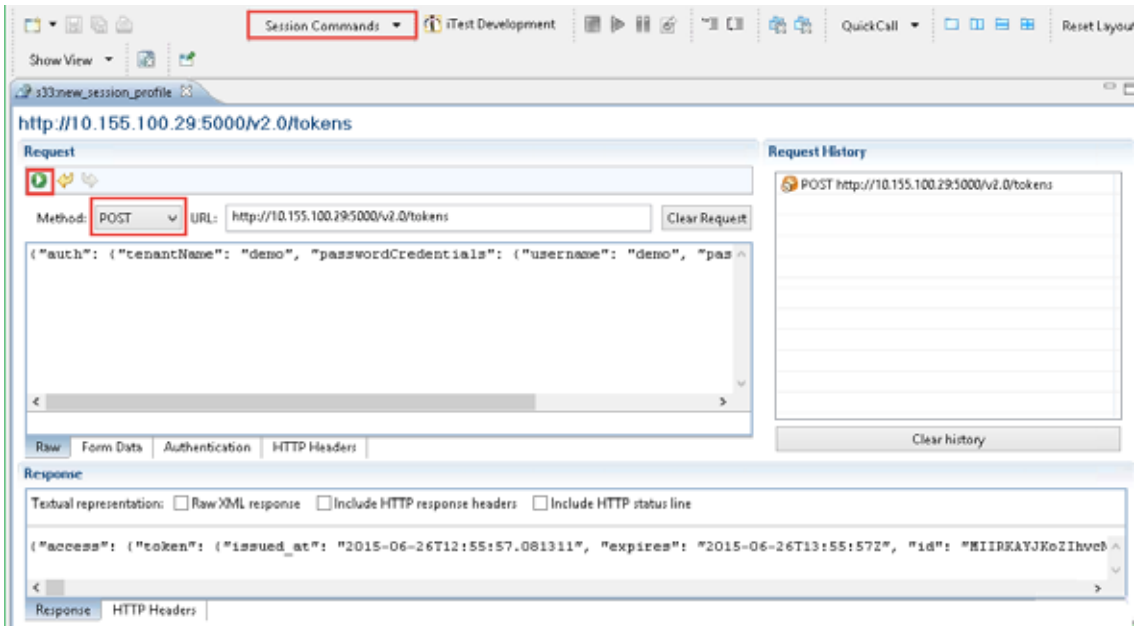
Step 1 Create a new session profile

Create a new session profile, notice that the new session type listed (e.g., Custom_session).



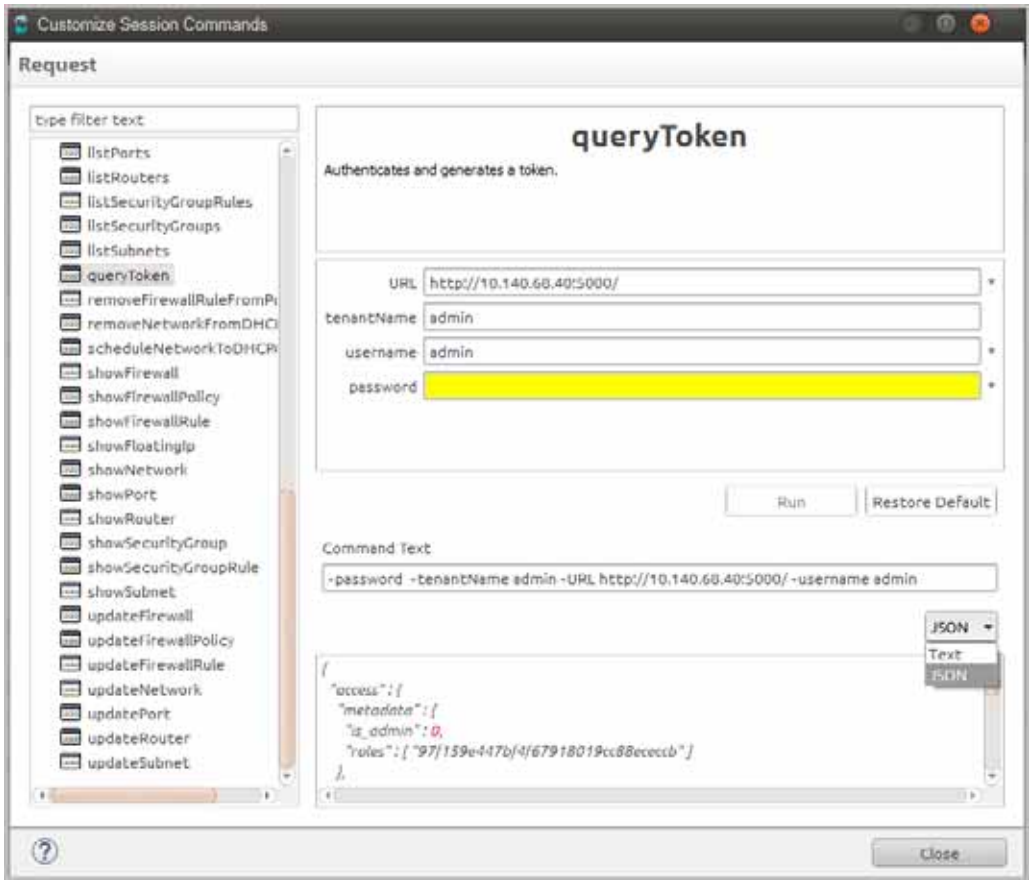
Step 2 Save and Start session

Click **Method: POST** (for example) to display the native commands. Select the required command and execute the base/native command (click the Green arrow).



On the top of the window a **Session Command** button appears. Click the down-arrow to view the list of commands and click Session Command to open the Customize Sessions Command window.

Filter the list and get to the required command (e.g., queryToken). Enter the details and click **Run**.



On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form.

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.
- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

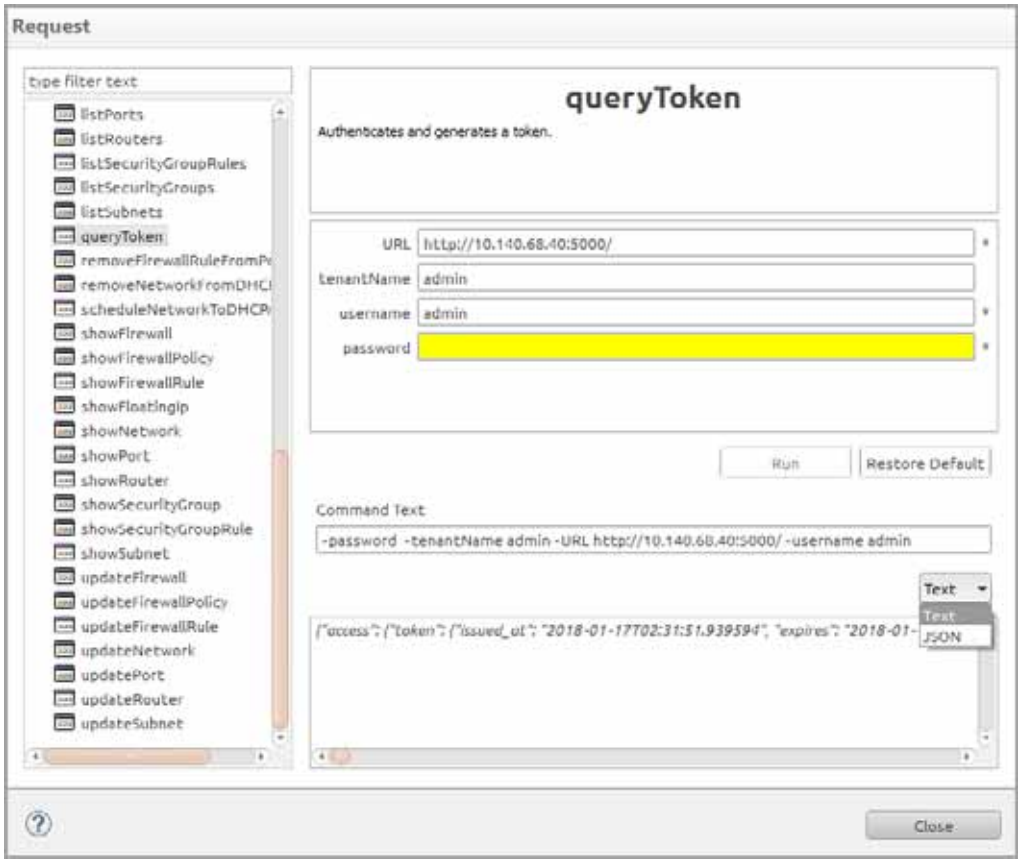
Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").

Note You may also fill in the parameters or enter the Command Text.

Click **Restore Defaults** to restore to the default values of command properties.

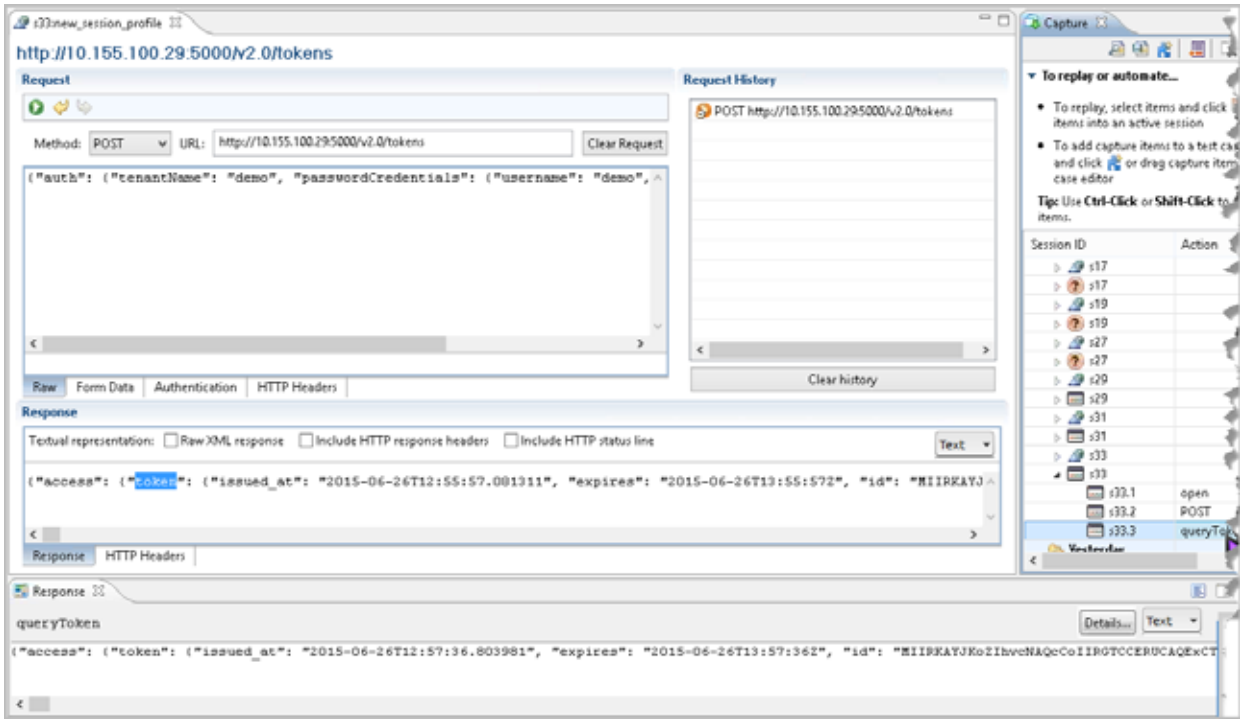
The illustration below shows execution (when you click **Run**) of the customized **queryToken** command and it's response.



Important The customized command parameters gets saved automatically when they are run.

Step 3 Executed Session Commands

All session commands executed, are captured in the Capture View as illustrated. You execute/replay these capture commands.



See also topic “Verify the customized session type” on page 775 to see how the custom session type are used.

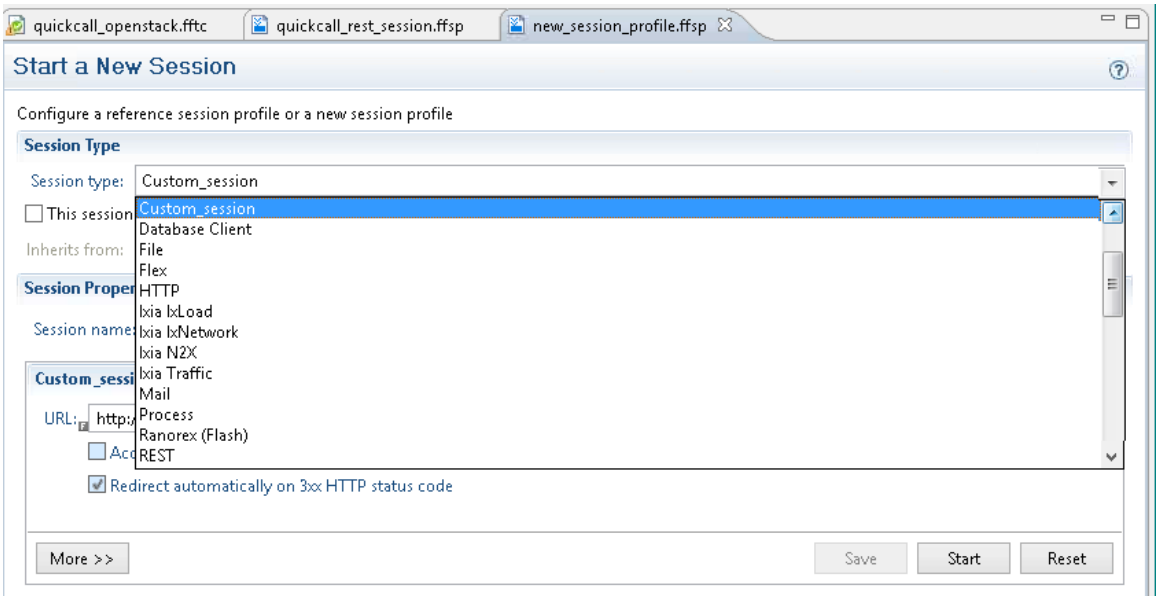
Using console based custom session

The console based custom command hides the base session from the your end user, and allows you to run the session commands via command line interface.

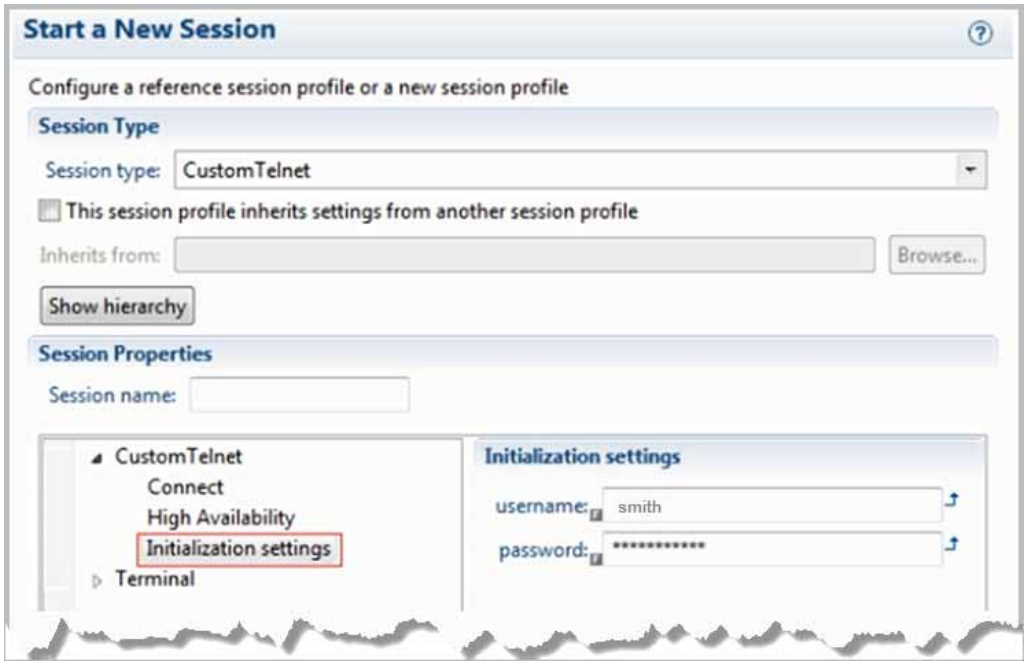
These steps describe opening the new custom session you built with the **Extends the existing session** *not selected* on page 762 (Step 4 “Define New Session Type wizard—Complete information and export session” on page 762) and executing the session commands and customizing them.

Step 1 Create a new session profile

Create a new session profile, notice that the new session type listed (e.g., Custom_session).



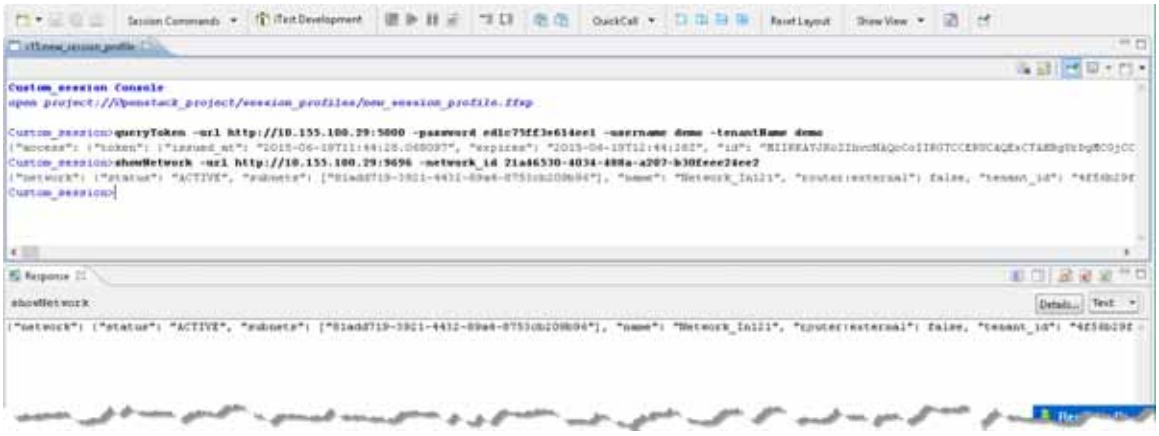
If you specified an initial procedure when building the custom session (**Session Initialization**, page 762), then a session profile page displays for you to update/input value for arguments defined in that initial QuickCall procedure. For example, a Telnet session that requires you to interact with terminal console and provide the username/password to login to the Telnet server is built into the quick-call (that is, enter the usernmae and password to login to the Telnet server).



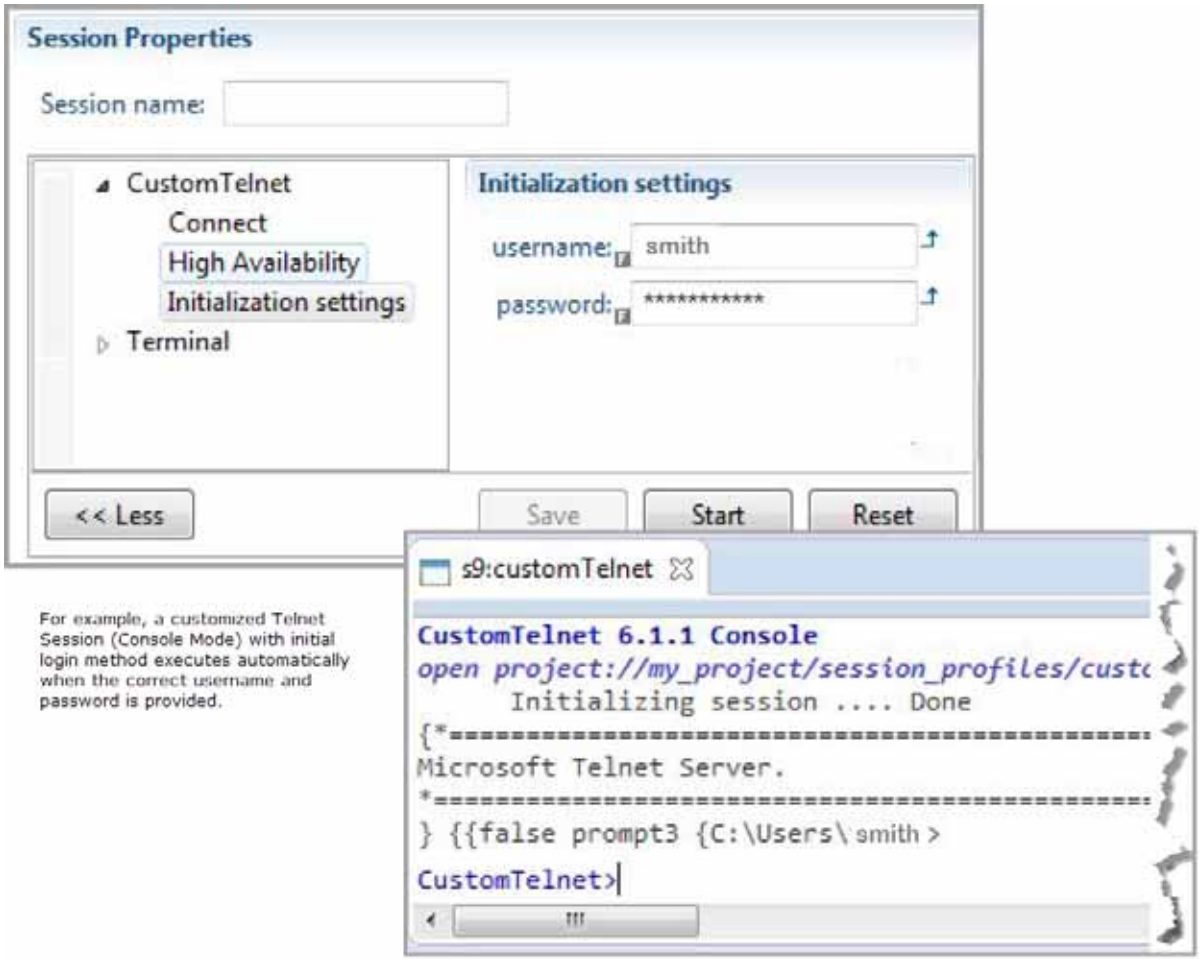
You may add values or update the existing values (in the arguments) as required.

Step 2 Start session

Enter the command and parameters as illustrated below. The session command executes and illustrated in the Response View.



If the session includes initialization settings, when the session starts, iTest will execute the initial method automatically and include its response in the open step's response and console (if running with capture mode). .



For example, a customized Telnet Session (Console Mode) with initial login method executes automatically when the correct username and password is provided.

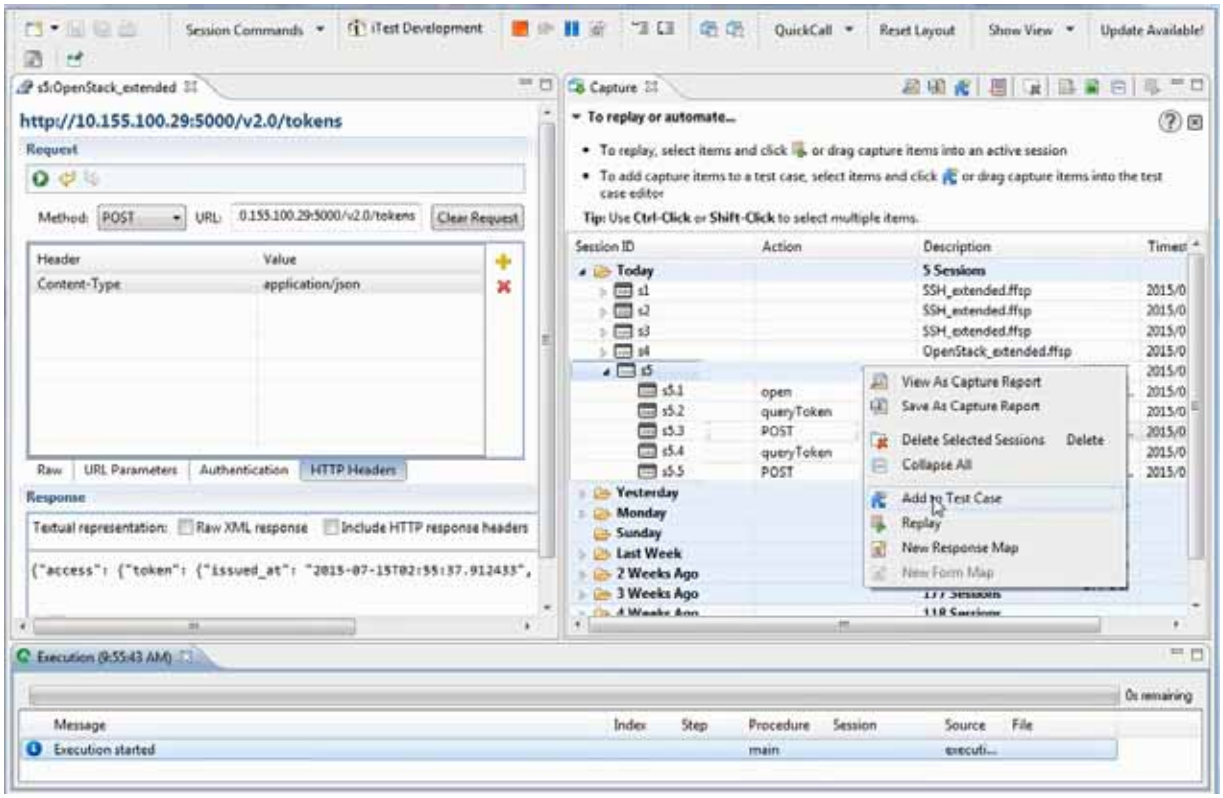
The executed commands may be viewed in the Capture View and used to generate a test case as described in “See also topic “Verify the customized session type” on page 775 to see how the custom session type are used.” on page 772. In addition, you may also add the saved commands with custom parameters to verify the custom session type as described in “Verify the customized session type” on page 775.

Verify the customized session type

Verify the customized session type by creating a test case, entering arguments in the test case editor, including the new customized session type and executing the test case. When you execute the test case, the custom session hides the implementation, and outputs the customized response, which helps with automation tasks.

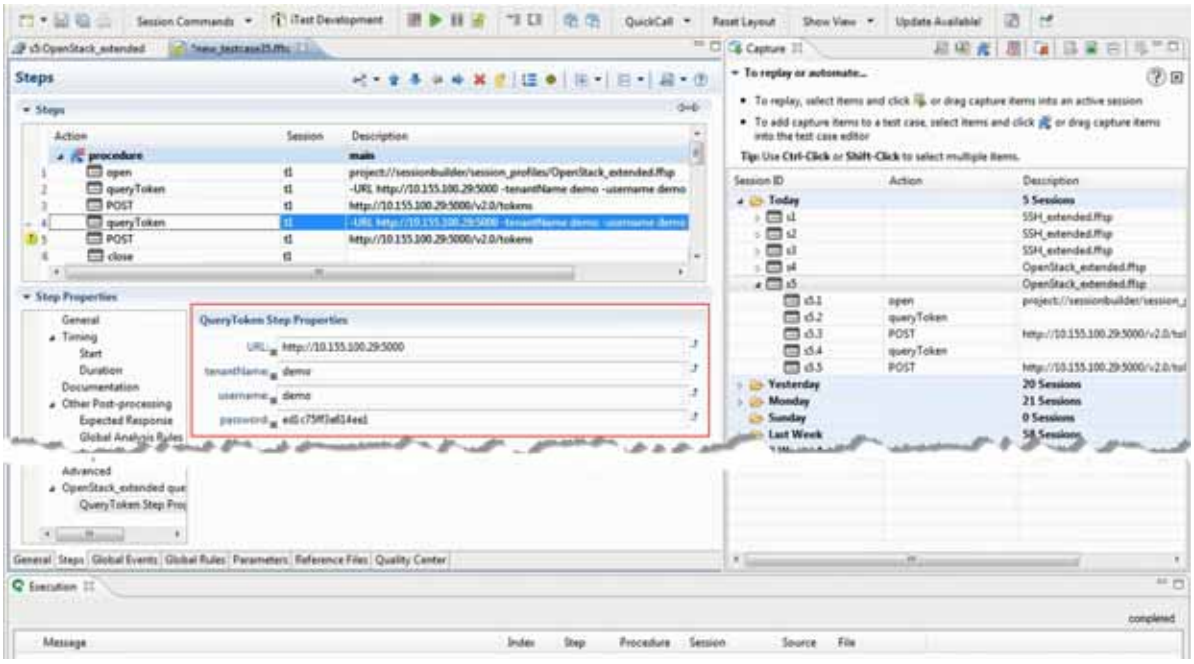
Generating a test case from capture view

The commands executed in the previous step are captured and display in the Capture View. Select the session, right-click, and add to a test case.

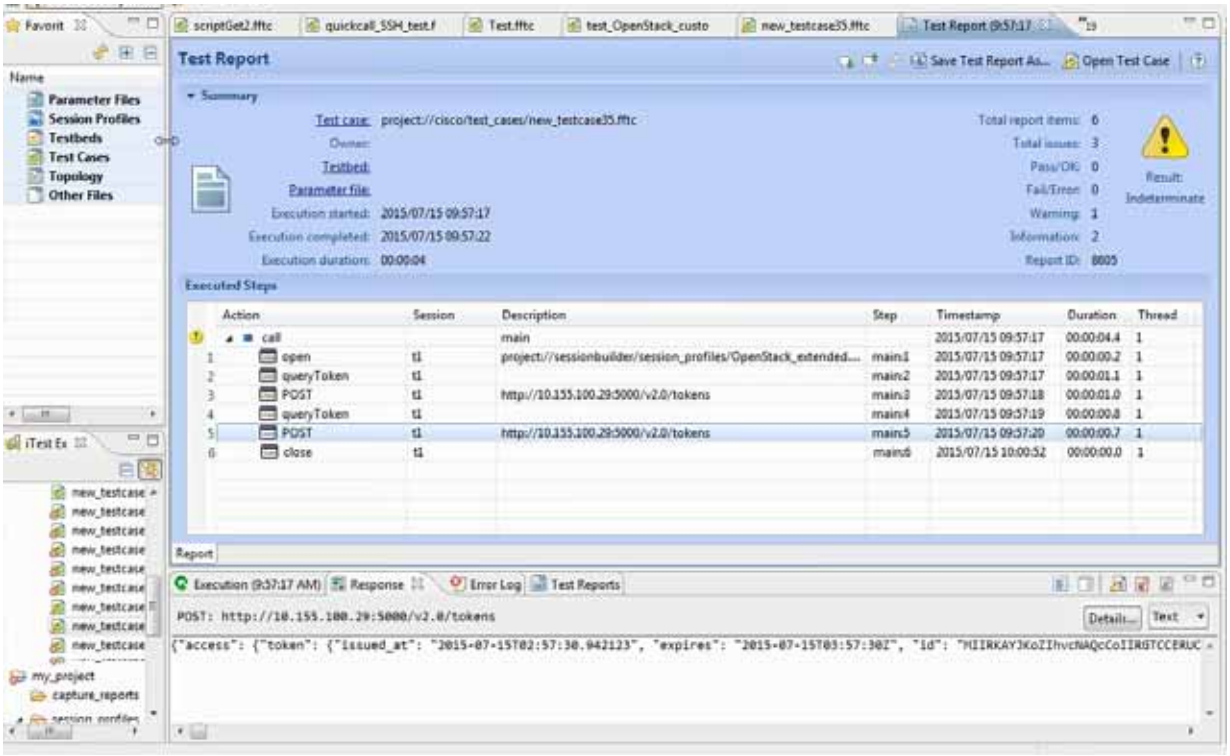


The selected session gets rendered into a new test case. Open the test case and view the step to see the custom value captured and displayed.

Note In custom sessions, the input argument value in Step properties mirror the Steps Description column.



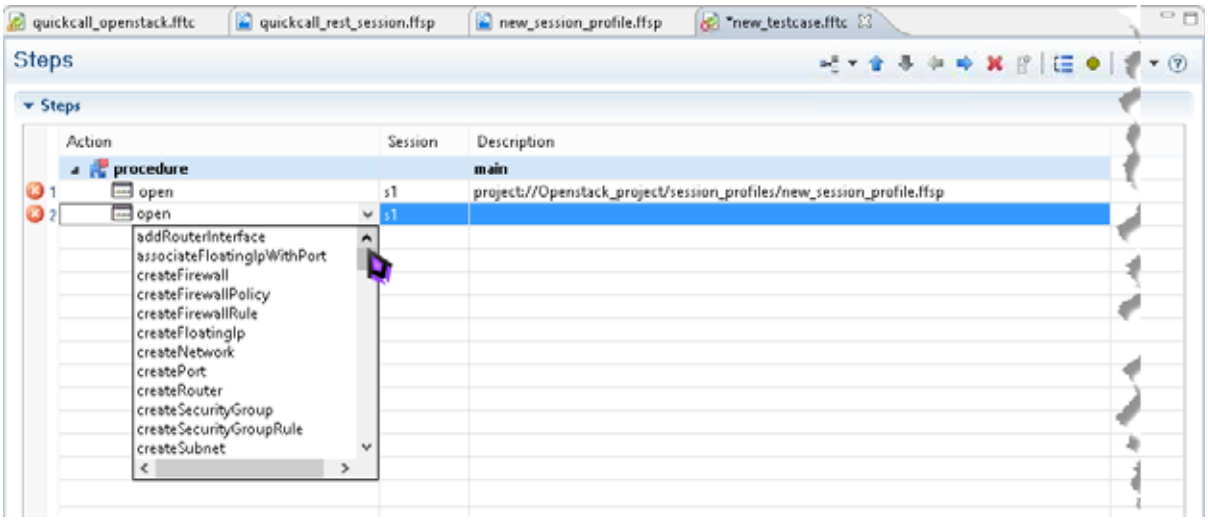
You may replay the test case and view results. Notice that the custom session hides the implementation associated with the native session, and outputs a customized response instead of, for example, the original REST response in case of OpenStack Neutron sessions.



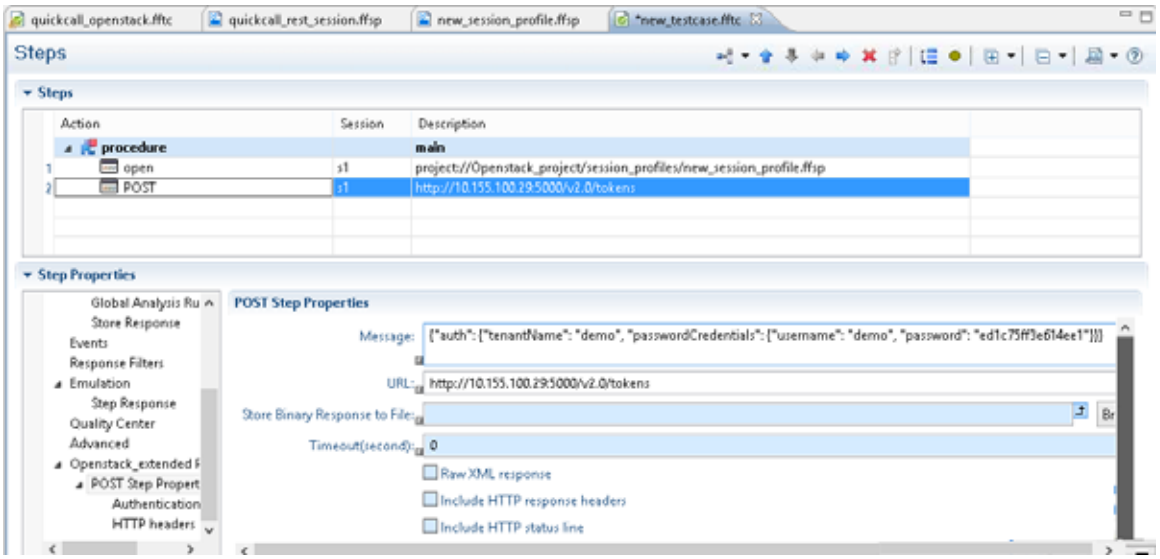
Creating a test case manually for verification

You may also manually create a test case with the customized command sessions.

Create a test case, enter arguments in the test case editor and also include the new customized session type.



All session commands display (above) and when you select to include the **queryToken** command, the command with the custom parameters includes.



You may execute the test case and view test result as described in the previous steps.

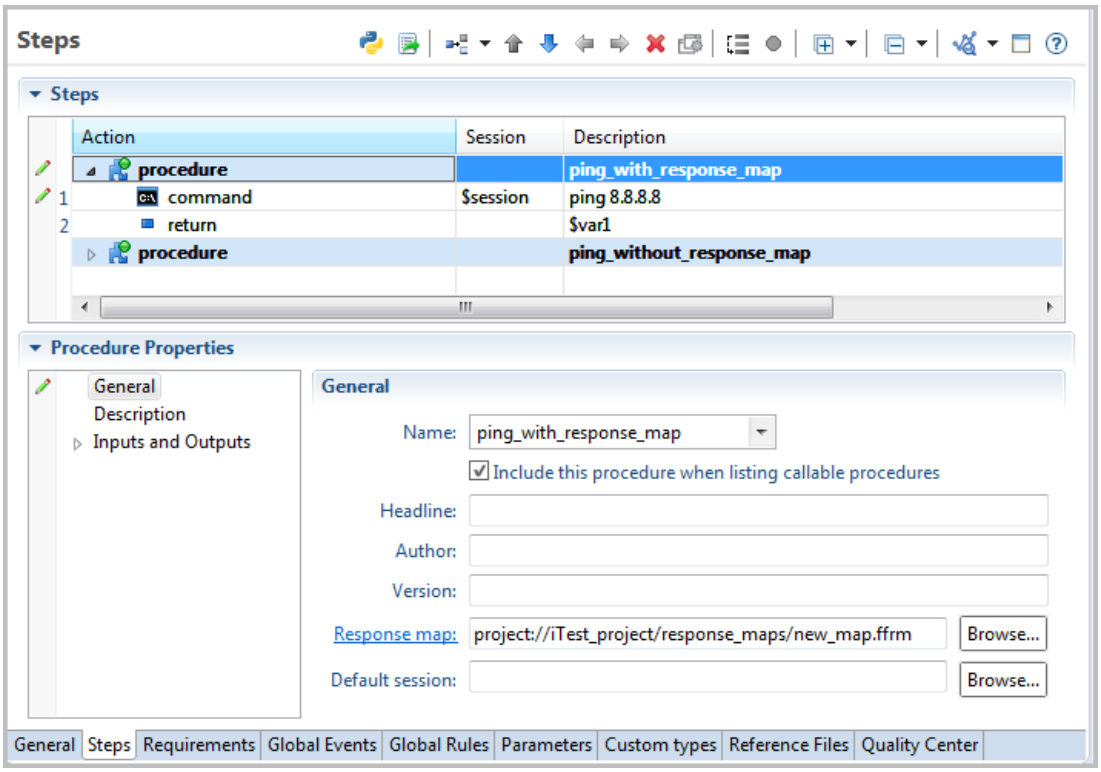
Verify Custom Session with response map

iTest exports any procedure-level response map defined in your QuickCall to a custom session. This allows furnishing the custom session with valuable queries and structures that returns output data with response mapped queries and structures.

The following illustrates an example QuickCall with procedure level response map that was exported as a custom session, a test case that uses the created custom session, and the response map applied to the output of the custom step.

Step 1 Example QuickCall with procedure-level response map

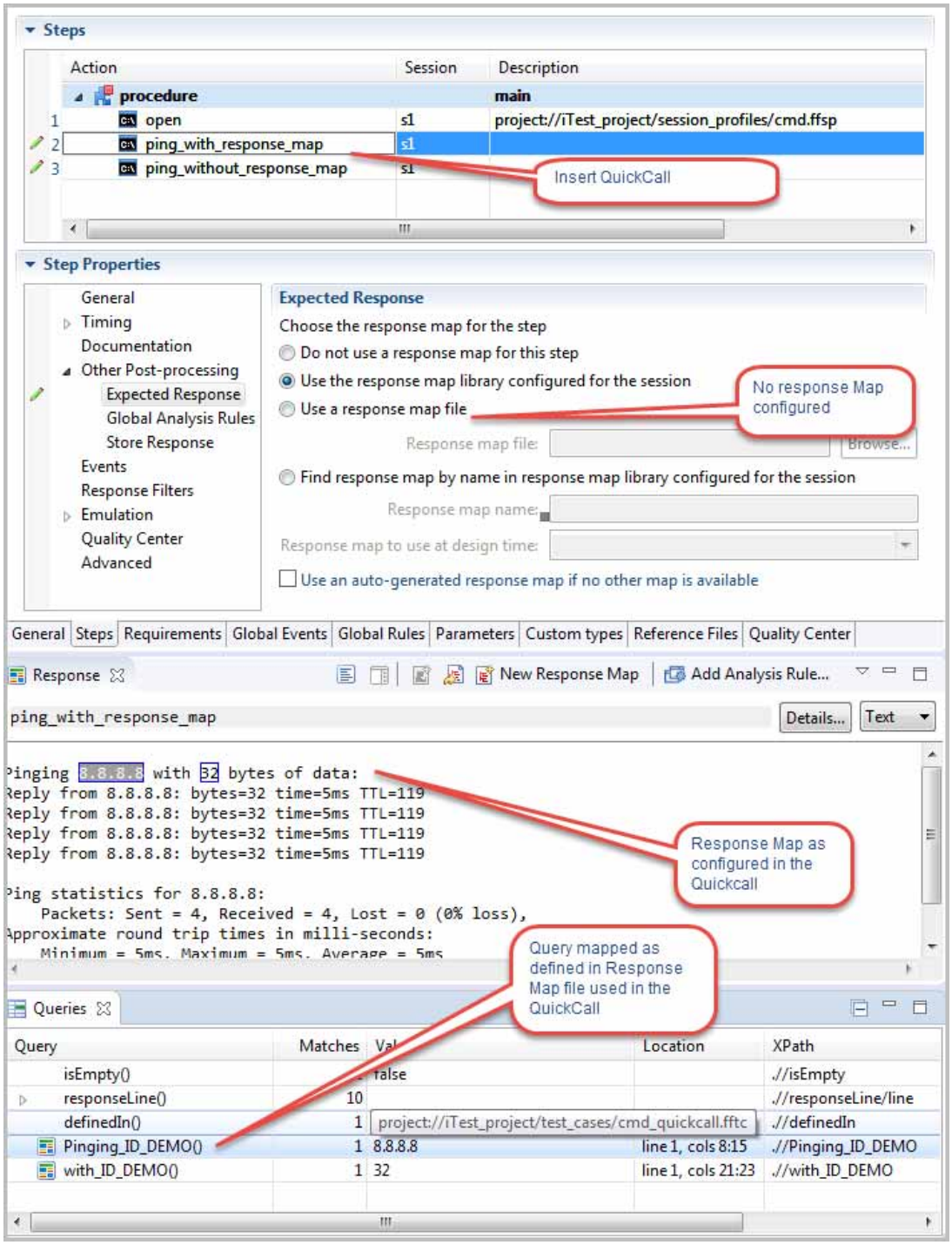
The response map file specified on the **Procedure Properties> General** page of a QuickCall.



Export QuickCall and create custom session as described in “Creating a custom session type” on page 759.

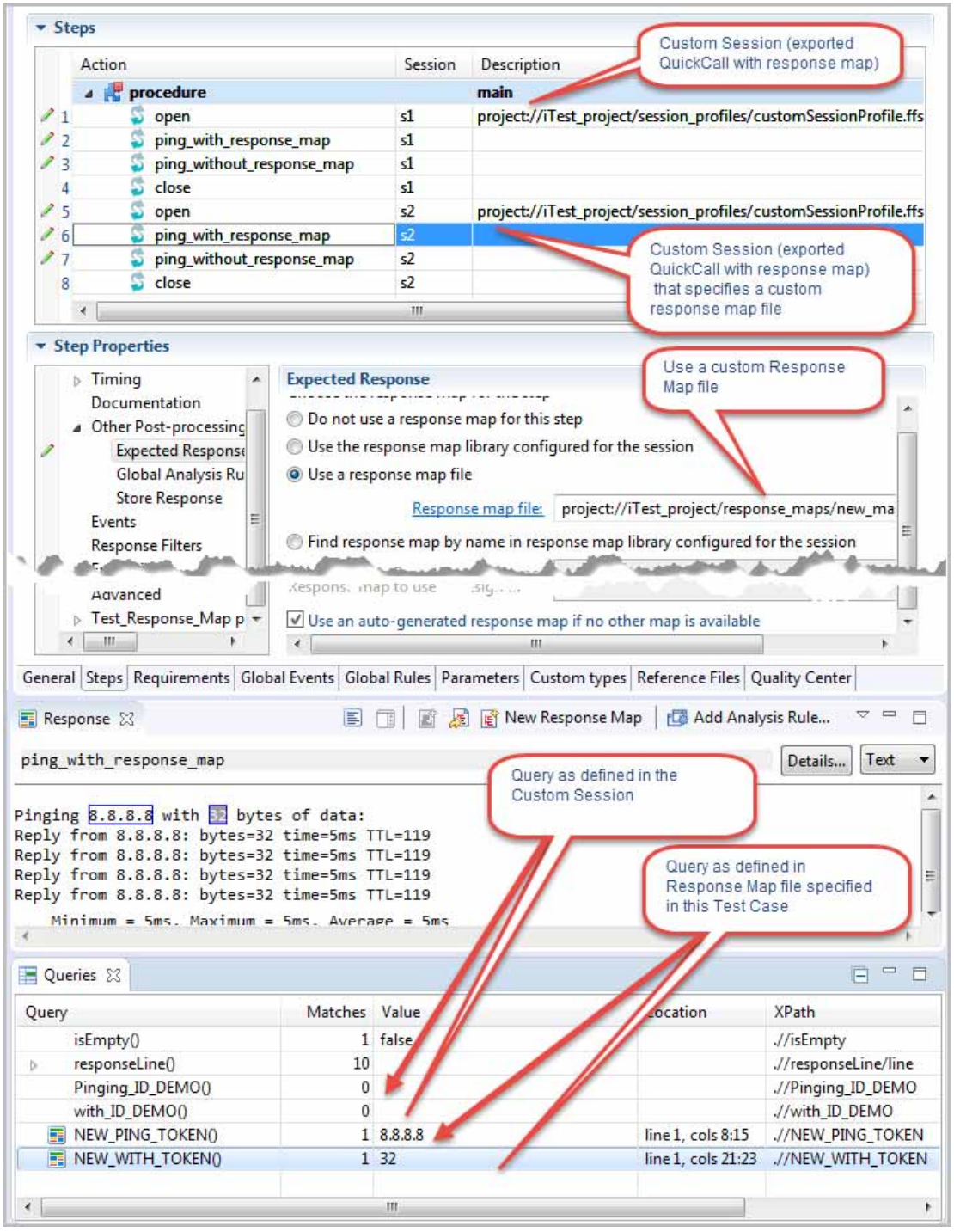
Step 2 Example QuickCall (not Custom Session) used in a Testcase

The example below shows the QuickCall used in a test case step and the response map applied to the test case step after test execution.



Step 3 Example Custom Session (exported QuickCall with response map) in a Test Case

The example below shows the Custom Session (with exported QuickCall with response map) used in a test case step and the response map applied to the test case step after test execution.



The same blue boxes that appear in the response window of a QuickCall step ("Example QuickCall (not Custom Session) used in a Testcase" on page 779), also appears in the response window of a custom session step (above). iTest ensures that all response map Pattern, Block,

and Table maps are applied to the custom session step responses (as configured at the QuickCall procedure level).

In addition to the response map file configured at the custom session level (exported QuickCall procedure level), iTest allows you to define custom response map file.

When a step has no response map defined, toggle **Use an auto-generated response map if no other map is available.** Unselect the option to display a blank Queries view or select option to display an auto-generated response map.

See "Expected Response" on page 180 (Chapter 8, "Test Case Editor").

Manually Installing Custom Session Type within iTest

Use the steps described in this section to configure and manage custom sessions as follows.

- "Install custom session exported to a local directory" on page 781.

  You may also use "Install custom session via iTest GUI Import Wizard" on page 783 instead of the above step.

- "Install custom sessions distributed through a Central Server" on page 782.

Install custom session exported to a local directory

Use these steps to install custom sessions on a local directory copied using the **Export to directory** option when building custom session (Step 2 "Select QuickCall Libraries and location to save" on page 760).

- Go to the **Help > Find and install new session types** menu on the top of the iTest window. The **iTest Session Settings** window displays with two tabs: **Available Session Types** and **Installed Session Types**.

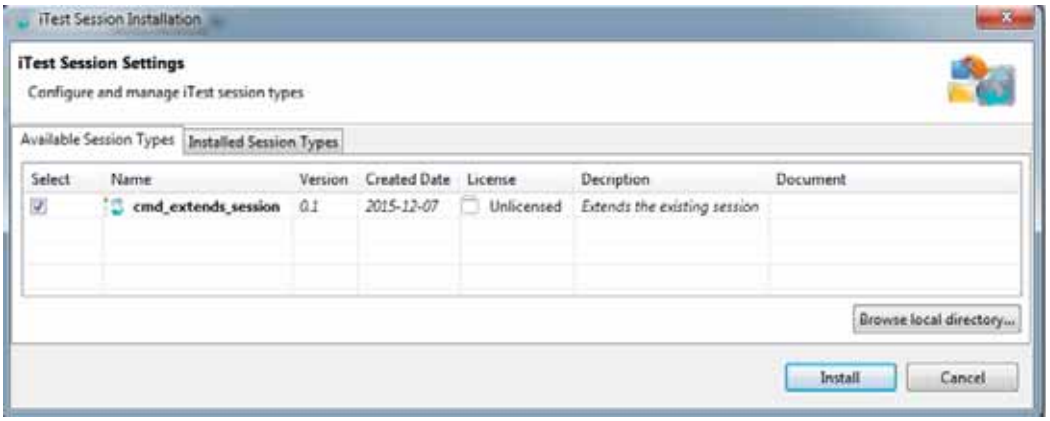
  Note You may get to the **iTest Session Settings** window via **iTest > Tools > Browse on iTest store**. See "Install custom session type using iTest > Tools > Browse on iTest Store" on page 784

- **Available Session Types**:

  Note The list is retrieved and populated from a Central Server (see "Install custom sessions distributed through a Central Server" on page 782).

- Click **Browse local directory**, navigate to location of the custom session files, select the folder and click **OK**.

  The **Available Session Types** tab get populated with the custom sessions in the folder. The list shows the name of the session type, version, date created, whether licensed, description of the session, and whether includes document, if any



- Select the Custom session file and click **Install.**

  iTest displays a message asking you to confirm installation of the session type. When you confirm, iTest informs you that it requires to restart in order to apply the new session type and asks for your confirmation again.

  iTest restarts and the installs the selected session (s), which will be available in the list of **Session Types** in the **Start a new session** window.

- **Installed Session Type**: Lists all the installed session types. You may uninstall session types as required. Uninstall also requires iTest to be restarted in order to apply the changes.

Install custom sessions distributed through a Central Server

Sessions built with the Session Builder may be deployed on a central server and distributed to your end-users by setting up the Update Site.

Step 1 Set up Central Site for software distribution

Go to **Windows -> Preferences** and set up the central site for software distribution as described in "Preferences: Spirent > Software Update" on page 866 (Chapter 41, "Configuring iTest Preferences").

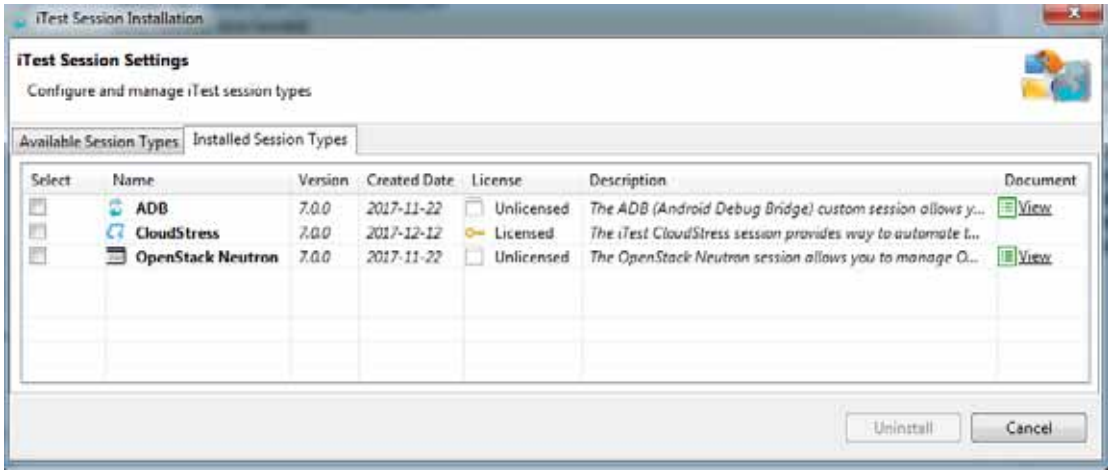
Step 2 Install new session types from the central site

Select menu **Help > Find and install new session types** on top of the iTest window. The iTest Session Settings display with two tabs: **Available Session Types** and **Installed Session Types**.

> **Note** You may get to the **iTest Session Settings** window via **iTest > Tools > Browse on iTest store**. See "Install custom session type using iTest > Tools > Browse on iTest Store" on page 784

- **Available Session Types**:

  The list is retrieved and populated from a Central Server. The list shows the name of the session type, version, date created, whether licensed, description of the session, and whether includes document, if any.



  - Select the session type (s) to install and click **Install**.
  - Click **Yes** when a message displays saying that iTest needs to restart in order to apply new session type, whether you to restart iTest.

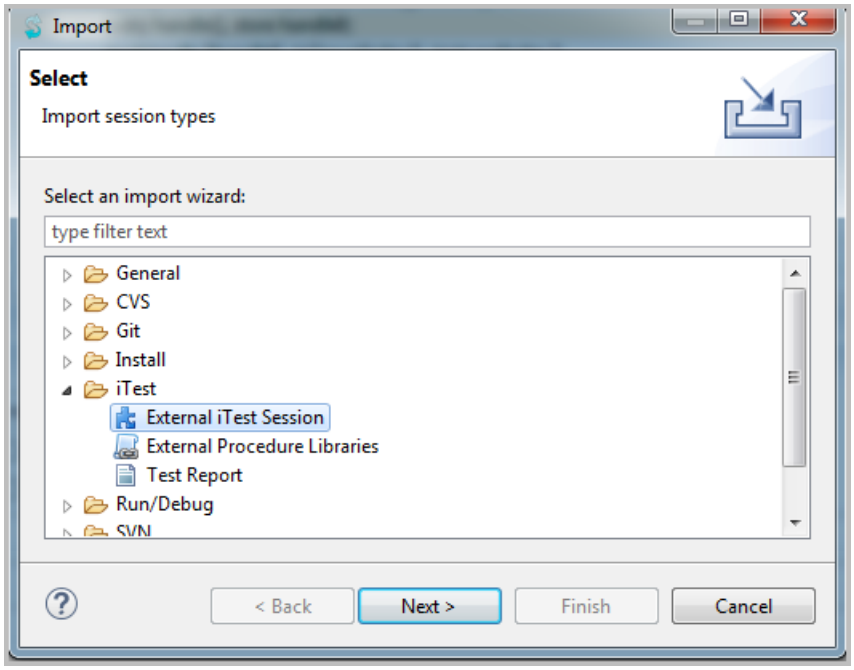
    iTest restarts and the installs the selected session (s), which will be available in the list of **Session Types** in the **Start a new session** window.

- **Installed Session Type**: Lists all the installed session types. You may uninstall session types as required. Uninstall also requires iTest to be restarted in order to apply the changes.

Install custom session via iTest GUI Import Wizard

Use these steps to install custom sessions distributed via email and/or copied to a local directory by using the **Export to directory** option when building a custom session (Step 2 “Select QuickCall Libraries and location to save” on page 760).

- Go iTest GUI Import Wizard: by clicking **File > Import** menu or right-click on iTest Explorer view and select **Open Import wizard**.



- Select **iTest -> External iTest Session** and click **Next**.
- Navigate to the custom session file (.ffst) and click **Finish**.
- Click **Yes** when a message displays saying that iTest needs to restart in order to apply new session type, whether you to restart iTest.

  iTest restarts and the installs the selected session (s), which will be available in the list of **Session Types** in the **Start a new session** window.

Install custom session type using iTest > Tools > Browse on iTest Store

In iTest go to the i**Test > Tools > Browse on iTest Store** menu on the top of the iTest window. The **iTest Session Settings** window opens (described above) you can Configure and manage iTest session types as required.

Export a QuickCall to Robot Library

Overview

iTest export wizard creates a keyword library that allows a Robot Framework user to build robot test cases and test suites constructed with keywords that map to QuickCalls. These test case and test suites (constructed with keywords that map to QuickCalls) allows developers to leverage useful iTest QuickCalls and Response Maps in their robot tests.

The exported keywords library is a Python file, generated from QuickCall (.fftc) file. The definition of keywords follows the syntax of Robot Static library APIs as document: Robot test library APIs (http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#different-test-library-apis)

Note Robot Framework is a Python-based, extensible keyword-driven test automation framework for end-to-end acceptance testing and acceptance-test-driven development. See http://robotframework.org/ for details.

The keyword library depends on Spirent Automation Libraries to invoke QuickCalls from iTest session. Ensure that you have configured Python environment.

Python Spirent Automation Library as described in Chapter 59, "Python Automation Library"

Note The exported Robot library supports Python 2.7 and Python 3.6.

iTest keyword libraries are compatible with Robot Framework IDE (RIDE), the integrated development environment to implement automated tests for the Robot Framework.

Note Using the iTest exported keywords and the support file in your Robot Framework test consumes iTest license.

This chapter includes the following topics that describes Robot key development using iTest, that is, exporting an existing QuickCall library as a Robot library, viewing the contents to understand how iTest maps the QuickCall procedures and keywords before using it in the Robot Framework test (developing Robot Script for using and executing in your environment).

Note Any changes to a QuickCall (except argument changes) that has been exported to a keyword file, does not have to be exported again as the change affects only iTest QuickCalls.

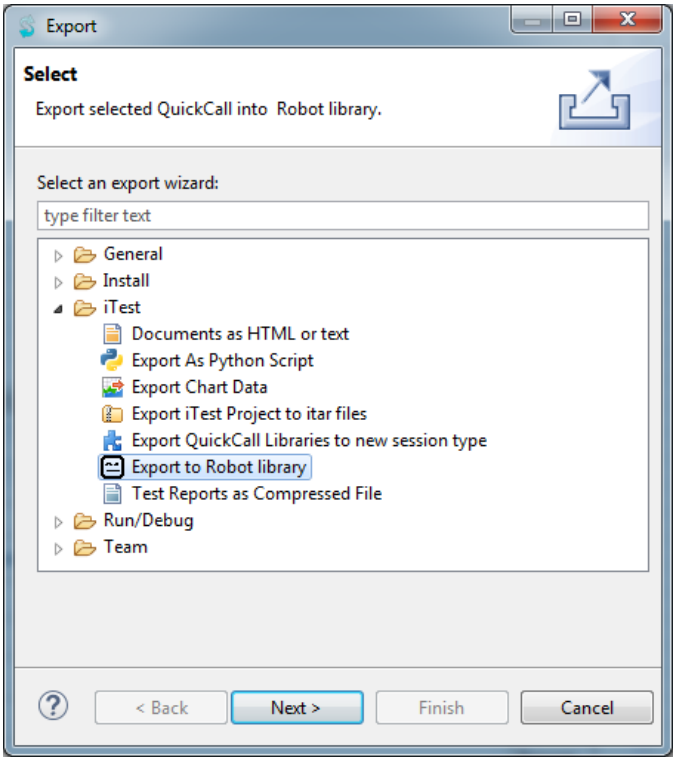
See "Session Profiles" on page 71 and "QuickCalls: Defining and using a library of custom actions" on page 205 for details about creating iTest Sessions and defining QuickCall libraries.

iTest Export Wizard—Export to Robot library

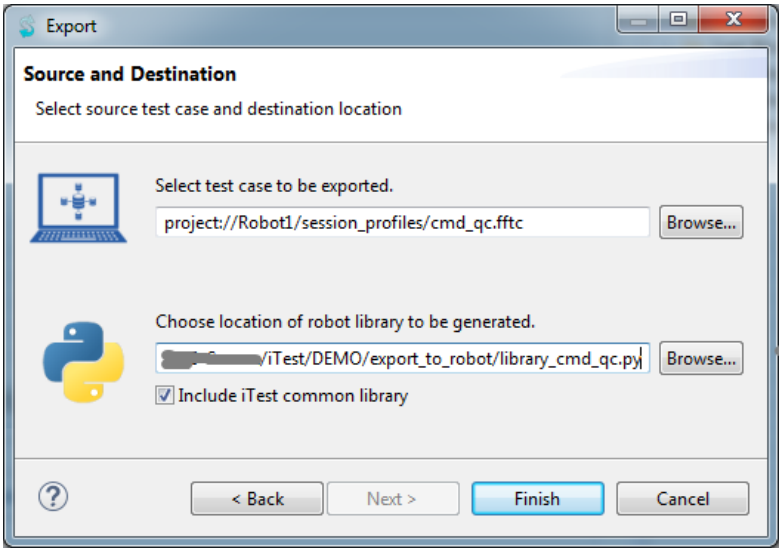
iTest Export Wizard exports any QuickCall library file (.fftc) to a Robot keyword library and includes QuickCall descriptions in the exported keyword library.

1 In the iTest explorer, right-click the document and click **Export**. The Export wizard opens.

2 On the **Select** page, select **iTest > Export to Robot library**. Click **Next**.



3 On the **Source and Destination** window, select source QuickCall to convert and the location where the generate Python Script will be saved.



- **Select test case to be exported:** Click Browse. Default location is Workspace. Navigate your workspace or the file system and select the file to be exported as Python Script and click **OK**.

  You may select only one QuickCall at a time and export to Robot Keyword file.

- **Chose location where robot library will be generated:** Click Browse. Default location is **file://C:/Users/user-01/** and the default name is the **QuickCall file name.py**. Browse to a location of your choice and click **OK**.

- **Include iTest common library:** Indicates whether the iTest common keyword file should be generated to the robot library file folder. This option is selected by default.

  When selected: The export wizard will include the iTest common keyword file—**iTestCommon.py**, in the robot library file folder.

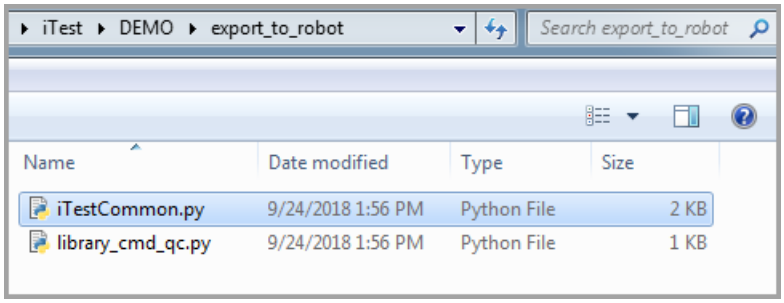
  This file is a support file required to use iTest exported Robot Keyword file in Robot script and execute in an external environment. You cannot execute QuickCall library without this file.

  When not selected: The export wizard will not generate the iTest common keyword file—**iTestCommon.py**.

4 Click **Finish** to convert the selected QuickCall to Robot framework file or click **Cancel** to discard the export operation.

iTest generates a Python file, example: **library_<quickcall>.py**, which a robot framework user can import it to robot test case.

If you had selected to **Include iTest common library**, iTest will generate an additional python file—**iTestcommon.py**. See the example below.



If the file exists, iTest displays a dialog asking you to confirm whether you wish to overwrite the existing file or change file name.

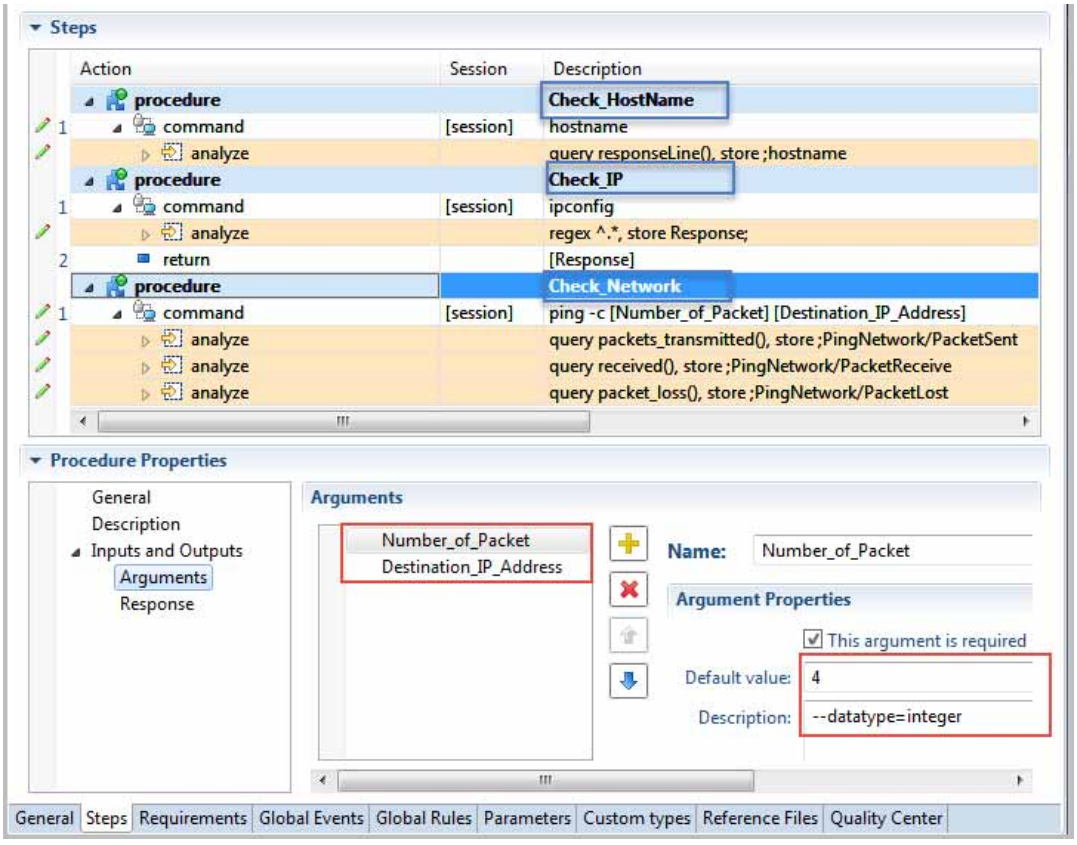
Contents of the exported Robot library

iTest Export Wizard exports any QuickCall library file (.fftc) to a Robot keyword library. The iTest exported keyword libraries are compatible with RIDE, the Robot IDE.

All procedure names of the QuickCall are exported as keywords in exported QuickCall library.The example below shows an existing QuickCall that was exported to Robot library with the keywords, arguments, default argument values, description of QuickCall, and any response defined.

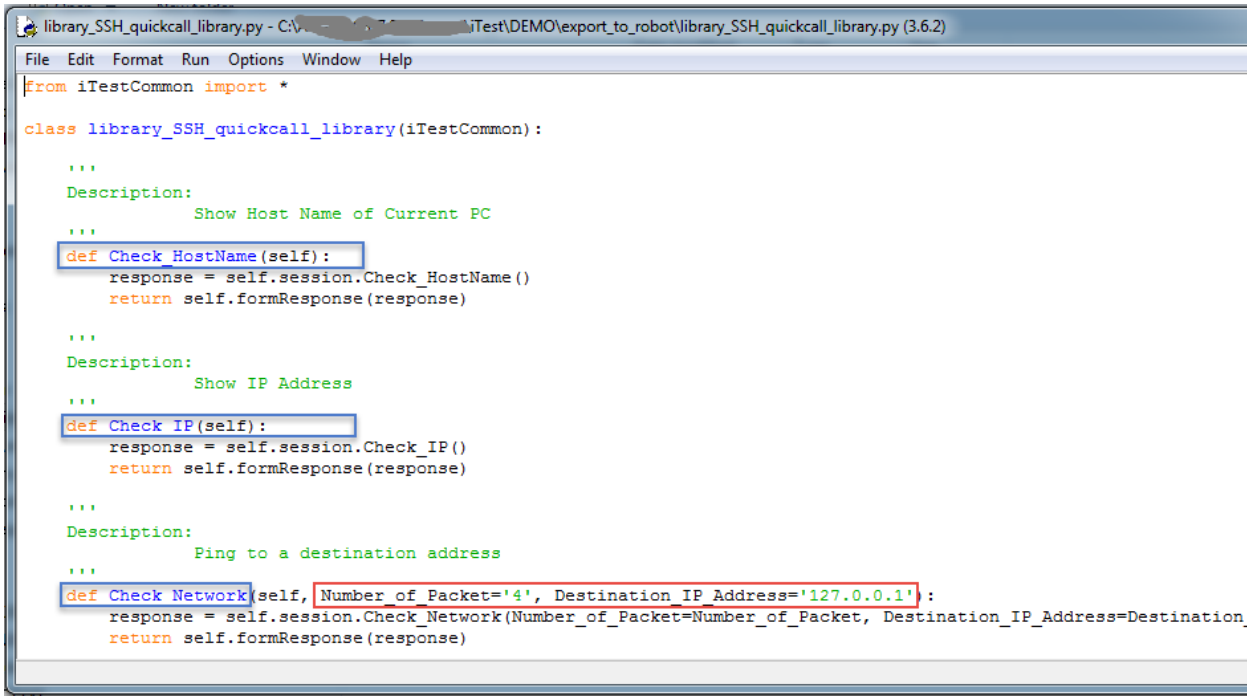
Step 1 A sample QuickCall to be exported

The diagram below highlights the procedure names (in blue) and the associated description, arguments, and the default value (in red) that will be exported as Robot dictionary keywords.



Step 2 A sample exported QuickCall to Robot Library (Python) file

The diagram below highlights the keyword definitions (in blue) that match the QuickCall procedure names and the associated description, arguments, and the default value (in red) exported to Robot library file.



All keyword library responses are dictionary structures.

- For QuickCalls that return a JSON response, that exact JSON response is returned by the keyword, and the Robot test case can validate contents using the Robot "Collections" library.
- For QuickCalls that do not return a JSON response, the raw text response from the QuickCall is made available in the dictionary under the key "text".

In addition, all queries related to the QuickCall's response are also inserted into the response dictionary structure.

Exported keywords in iTestCommon.py

The i**TestCommon.py** is a support file that executes the iTest exported key words file. This sections lists the keywords in **iTestCommon.py** file.

Important You cannot run QuickCall library without this file.

iTest creates keywords in **iTestCommon.py** that ensures a Robot test case file can connect to iTest, open project, start session, switch session (in case of multiple sessions), close sessions, and display any response and queries set up.

The example below is an extract of **iTestCommon.py** file.

```
from SpirentSLC import SLC
from SpirentSLC.Execution import *

import re
import io
import logging
import collections
from time import sleep, time
from datetime import datetime

global session
global sessionMap
global sessionIndex
global index

class iTestCommon(object):

	ROBOT_LIBRARY_SCOPE = 'GLOBAL'

	def connectItest(self, host='localhost:9005'):
		self.slc = SLC.init(host)

	def openProject(self, project='my_project'):
		self.project = self.slc.open(project)

	def startSession(self, name, alias=None):
		global session
		open_command = "self.project." + name + ".open()"
		if (alias is None):
			alias = name
		session = eval(open_command)
		self.sessionIndex[self.index] = alias
		self.sessionMap[alias] = session
		self.session = session
		self.index += 1
```

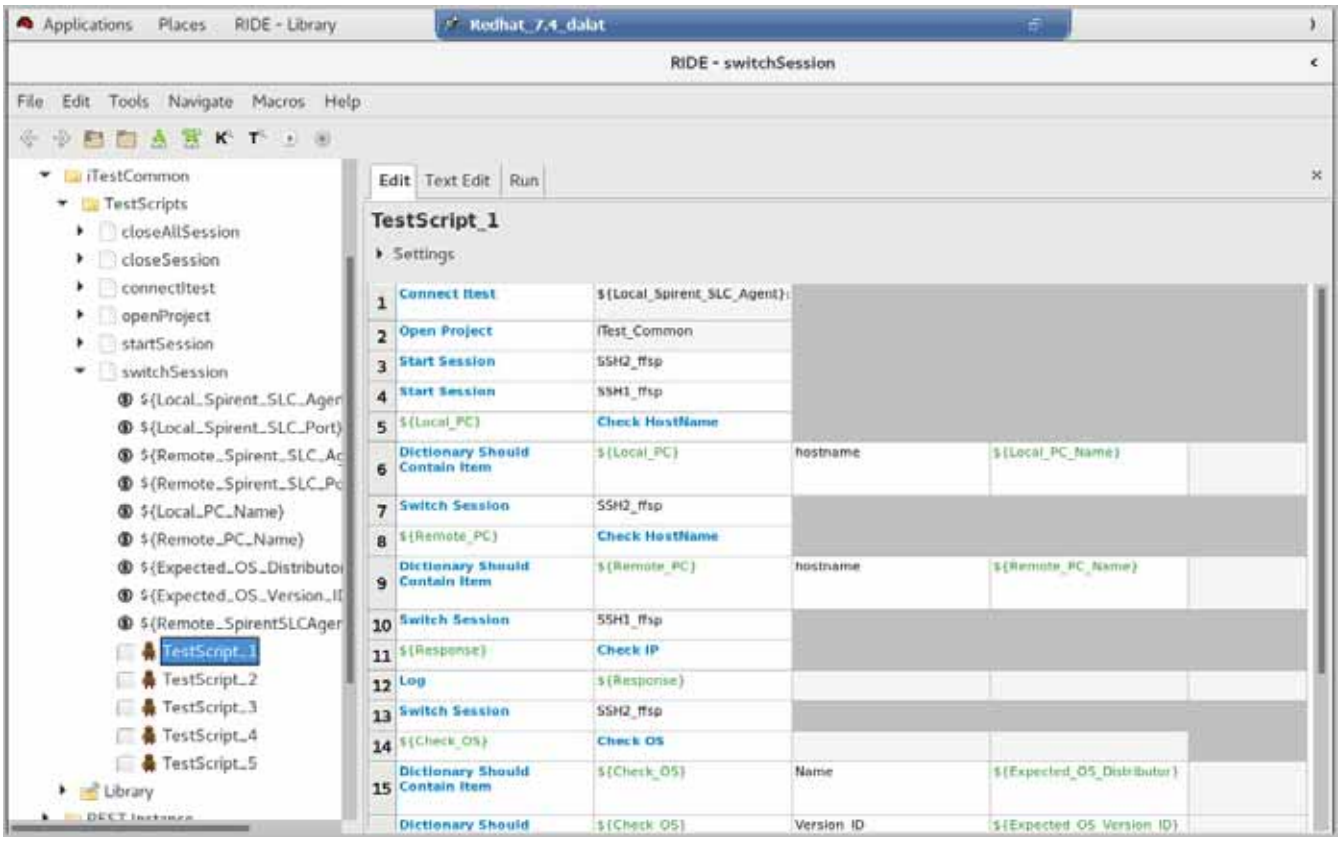
```
        def switchSession(self, index_or_alias):
                global session
                global index
                if self._representsInt(index_or_alias):
                        i = int(index_or_alias)
                        if (i <= 0 or i >= self.index):
                                raise Exception('Invalid index: ' + i)
                        index_or_alias = self.sessionIndex[i]
                try:
                        session = self.sessionMap[index_or_alias]
                        self.session = session
                except Exception:
                        raise Exception('Invalid alias: ' + index_or_alias)

        def closeSession(self):
                global session
                session.close()

        def closeAllSessions(self):
                global sessionMap
                global sessionIndex
                global index
                for key, ss in self.sessionMap.items():
                        ss.close()
                self.sessionMap = {}
                self.sessionIndex = {}
                self.index = 1

        def _formResponse(self, response):
                if response.json != None:
                        return response.json
                else:
                        responseDictionary = {'text' : response.text.strip()}
                        responseQueryList = eval(response.queries())
                        for key in responseQueryList:
                                responseDictionary[re.sub('[()]','',key)] =
eval('response.' + key)
                        return responseDictionary

        def __init__(self):
                self._description = 'common library'
                self.sessionMap = {}
                self.sessionIndex = {}
                self.index = 1


        def _getSession(self):
                global session
                return session

        def _representsInt(self, s):
                try:
                        int(s)
                        return True
                except ValueError:
                        return False
```

Execute exported Robot library from an external environment

You may create a Robot test script with the exported Robot library as required.

These steps provides an example of a Robot script with the exported Robot library and executing from an external environment.

Create Robot test case (e.g., in Robot Framework IDE). The diagram shows an example of iTest exported keywords file in RIDE.



1. Go to the folder that contains the exported Robot library file

   Example: C:/Users/<username>/Robot_Library/

2. Open Robot Framework IDE.

3. Import the exported Robot library and the support file

   Example: library_CMD_quickcall_library.py and iTestCommon.py

4. Create a Robot test case with default value:

   Library.../lib/library_CMD_quickcall_library.py

   Library Collections

   Test Cases

   ....

5. Save and Run the Robot test case in Robot Framework IDE.

The following shows an example Robot test case and the corresponding log file that includes keyword and output.

```
*** Settings ***
Library       library_ssh_1.py
Library       library_wireshark1.py
Library       iTestCommon.py
Library       Collections

*** Test Cases ***

My simple test
 Connect iTest
 Open project  iTest72
 Start session  ssh_1_ffsp  alias=ssh
 Start session  wireshark1_ffsp  alias=ws
 Switch session  ssh
 ${response}=  Get Interface  ens33
 Dictionary Should Contain Item  ${response}  mac  a4:4e:31:74:71:d4
 Switch session  ws
 Load Pcap  1.pcap
 ${response}=  Show Ds Field  2
 Dictionary Should Contain Item  ${response}  ipdsfield  0x00000000
 Switch session  ssh
 ${response}=  Get Cpu Info
 Dictionary Should Contain Item  ${response}  vendor_id  GenuineIntel
 Close Session
```

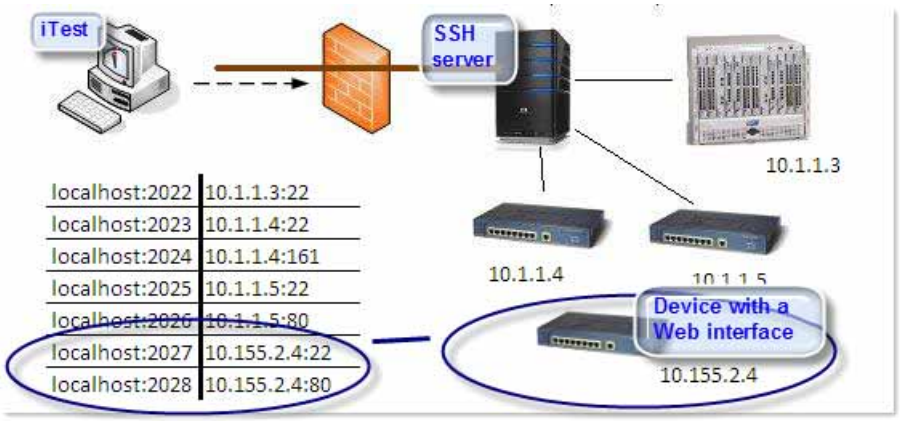
CHAPTER 35

Using SSH Local Port Forwarding to connect iTest from your desktop through a firewall to your lab devices

Using iTest over a VNC connection to try to connect to lab devices through the lab firewall is difficult (wrong screen dimensions, latency, no tooltips, and so on) and can require complex setup and troubleshooting. Instead, use iTest's SSH **Local Port Forwarding** capability for a very clean connection.

Example

Here is a typical situation: You want to use iTest from your desktop, but the lab is firewalled. In this example, we will set up a iTest session with a device in the lab (10.155.2.4) that uses both a Telnet interface and a Web interface. The table lists the ports that we have decided to use and will configure on the session profiles for both the SSH server and the Web device.



Step 1 Configure a session profile for the SSH server

1. Create a session profile for the SSH server. On the **SSH** properties page, specify the IP address of the SSH server (iTest can automatically assign an available port on the local host).
2. On the **More > Local port forwarding** properties page, configure the SSH session with a port forwarding list:

   Check the box to **Enable local port forwarding**.

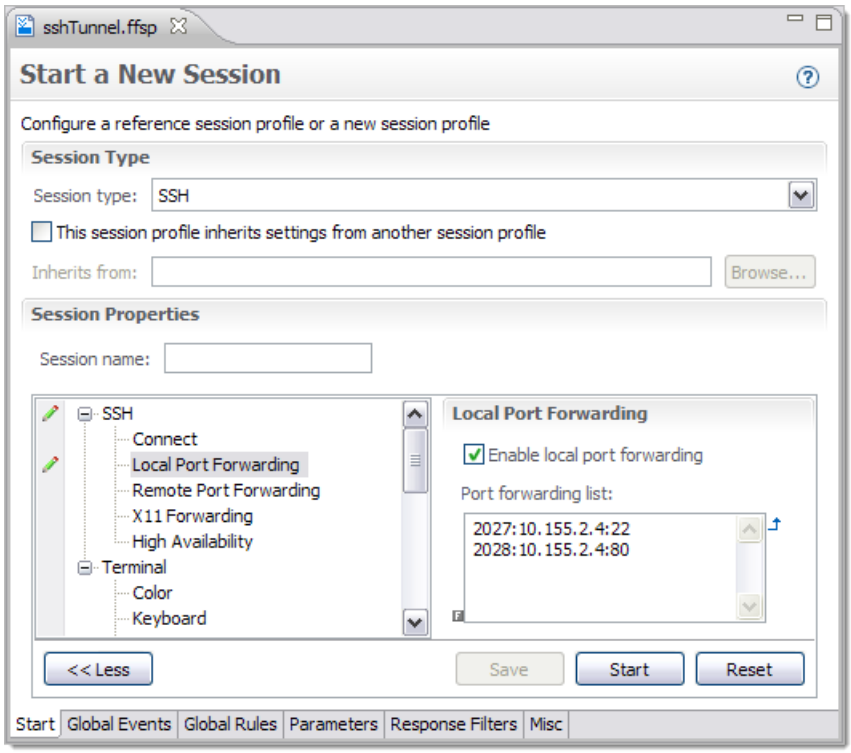
   Define a port for each device beyond the firewall that you will connect to. For each port forwarding pair, provide host and port information in the following format (one pair per line):

[localIPaddress:]localPort:remoteIPaddress_or_hostName:remotePort

Notice that **localIPaddress:** is optional.

Field substitutions are supported in any part of the text.

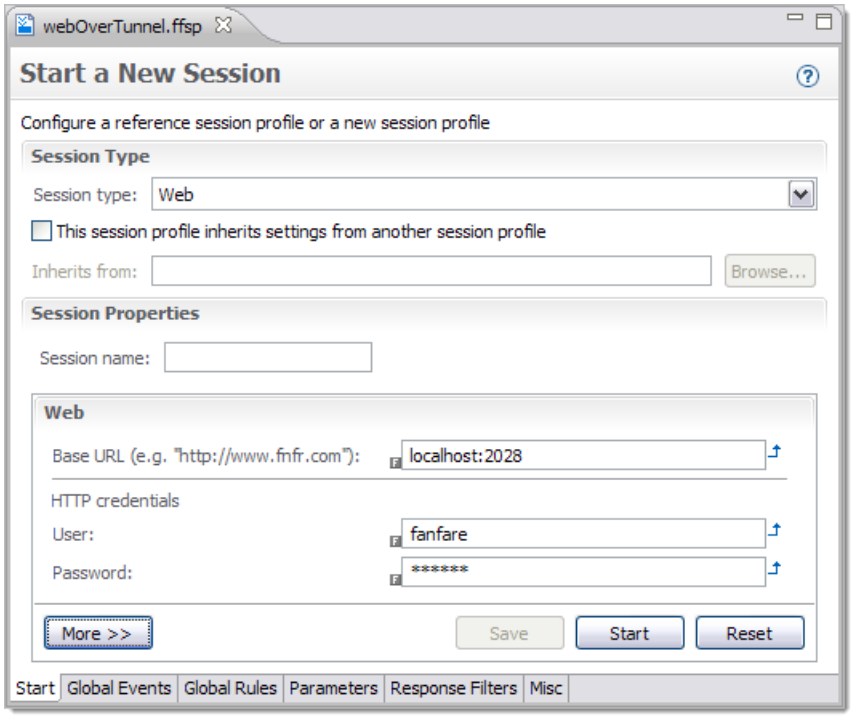
In our example, for the router at 10.155.2.4, we specify that traffic on port 2028 (from the SSH server that is communicating with iTest) should be forwarded to port 80 (typically HTTP traffic) on the router. We also specify that traffic on port 2027 should be forwarded to port 22 (Telnet)



Step 2 Configure a session profile for each device behind the firewall

1 In our example, we create a session profile for the router with the Web interface.

2 We specify the **Base URL** as the device's IP address and the port number that it uses for Web traffic with the SSH server.



CHAPTER 36

Making your test case thread-safe: Actions that help you to synchronize execution threads

Overview: Synchronizing threaded execution in iTest

iTest supports multi-threaded execution and provides the following types of tools for managing thread synchronization:

- **lock: Ensure that only one thread at a time can work within a critical set of steps**

  A **lock** step specifies a lock name, and all steps that are indented as children of the **lock** step are locked (often referred to as *mutex* or *semaphore*). The named lock is released for use by other **lock** steps when execution finishes for the currently locked steps. A lock is useful in the following situations:

  - When you need to ensure that no steps in a separate section of a test case or procedure alter a variable while a particular set of steps is working with it
  - To ensure that all steps in a procedure that opens a session, performs actions in the session, and finally closes the session can execute atomically — that is, with no steps in other threads changing something about the session.

- **signal: Pause one or more threads until one or more specified signals occur**

  The **signal** group of actions enable a variety of options:

  - Pause a thread until it is signaled to proceed by a specified event. For example, to encapsulate the procedure that opens a session so that no other steps in the session will proceed until the session is open
  - Pause particular threads of execution until other threads notify (signal) them that it is safe to proceed. This capability enables you to encapsulate test cases and to improve the multi-threaded aspects of test cases.
  - Pause threads until explicitly signaled. For example, you can begin a test case by initializing several devices and other steps or procedures cannot use the devices until initialization is complete.
  - Advanced users can combine synchronization actions to prevent deadlock, starvation, livelock, and other common liveness issues

- **waitThread: Ensure that execution does not continue until specified threads finish execution**

  A **waitThread** step completes only when the last thread finishes (you specify which threads to wait for). Execution then continues with the next step.

The Threads view displays synchronization information

To support you while you develop and debug test cases, the following columns in the Threads view display synchronization information:

Awaited signals	Displays the names of any signals that must be activated for the step to proceed. The signal can come from a **signal**, **signalAll**, or **signalActivate** step. For **signalWaitAll** steps, all of the signals in the list must be active before the step can proceed.
Owned locks	Displays the names of all locks that the step owns — locks that were set by the step or by a parent step.
Awaited lock	Displays the name of the lock that must be released for the step to proceed.

For more information on the Threads view, see “Threads view” on page 276.

Events that support you in debugging threads

See the descriptions for “OnThreadEnter” on page 617 and “OnThreadExit” on page 617.

Overview: Actions that help you to synchronize (lock) threads

Follow these guidelines in naming signal events:

- Must be alphanumeric and can include underscore characters
- Support field substitution.

Action	Command property value (in the **Description** cell)	Description
Note You can perform the following actions only as EXEC actions in steps (and not as Actions for Events).		
lock	*lockName*	Ensures that only one thread at a time is working inside the set of steps that is locked on *lockName*. See “lock: Ensure one thread for a specified block of code” on page 801.
signalWait	*eventName* [, *eventName*, ...]	Causes the currently executing thread to sleep until it is resumed by a call from **signal**, **signalAll**, or **signalActivate** with any one of the specified event names. See “signalWait: Sleep the currently executing thread” on page 802.

signalWaitAll	*eventName* [, *eventName, ...*]	Causes the currently executing thread to sleep until all specified events have been signaled or activated (by **signal**, **signalAll**, or **signalActivate**). See "signalWaitAll: Wait until all specified events have been signaled or activated" on page 803.
Note You can configure the following actions both as EXEC Actions in steps and as Actions defined for Events.		
signal	*eventName*	Wakes a thread that is waiting on *eventName* and causes it to continue execution. See "signal: Wake a thread that is waiting on an event" on page 804.
signalAll	*eventName*	Causes the currently executing thread to sleep until all specified events have been signaled or activated (**signal**, **signalAll**, or **signalActivate**). See "signalAll: Wake all threads that are waiting on an event" on page 804.
signalActivate	*eventName*	Turns on the event called *eventName*. See "signalActivate: Turn a signal on" on page 805.
signalClear	*eventName*	Removes any instances of the event named *eventName* that had previously been activated either by a **signalActivate** step or by a **signal** command. See "signalClear: Deactivate a signal" on page 805.

lock: Ensure one thread for a specified block of code

The **lock** action ensures that only one thread at a time is working inside the set of steps that is locked on the *lockName* that is specified for the **lock** step.

Action	**Command property value** (in the **Description** cell)
lock	*lockName*

Defining and using a lock

All of the steps that are indented under the **lock** step are included in the lock. The lock is released when all of the locked steps finish executing.

When a thread arrives at a step that is locked (for example, with lock name **lockA**) , it determines whether the lock is currently in use. Then:

- If the lock is not in use, the thread executes the step
- If the lock is in use, the thread waits until the lock is released
- If the thread owns **lockA**, it can enter any step that is locked on **lockA**. (This prevents deadlock if a locked step calls a procedure that has a step that is locked on the same lock.)

CAUTION You are allowed to start new threads of execution inside a locked block, but this practice can render the code ***not thread safe***. The new thread will own no locks and will proceed until either the thread ends or it hits another locked region.

Example

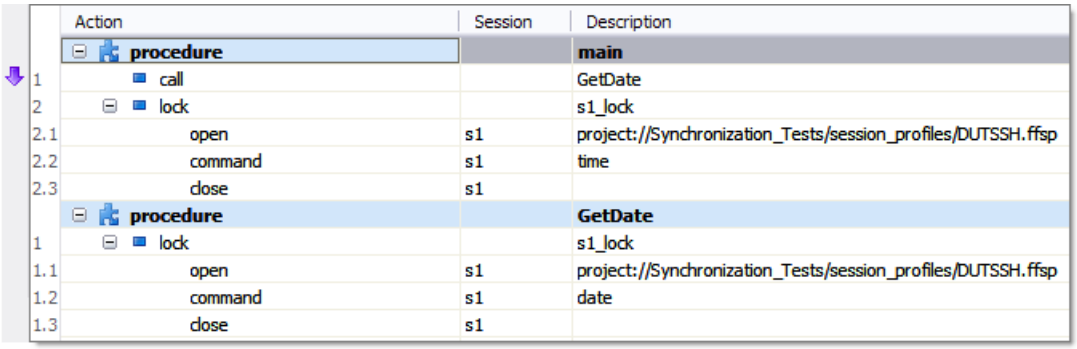
a The **main** procedure immediately calls the **GetDate** procedure.

b Step 1 of the **GetDate** procedure sets a lock called **s1_lock** and then proceeds with its execution.

c Meanwhile, as **main** moves on to step 2, it encounters a lock called **s1_lock**. Because **s1_lock** is currently in use, **main** cannot proceed to step 2.1 until the lock is released.

d The **GetDate** procedure continues and eventually **s1_lock** is released after step 1.3 (because step 1.3 is the last step that is indented under the **lock** in step 1 of the **GetDate** procedure).

This arrangement ensures that the **open** action in step 2.1 of **main** does not occur until the **GetDate** procedure has finished executing.



	Action	Session	Description
	procedure		**main**
1	call		GetDate
2	lock		s1_lock
2.1	open	s1	project://Synchronization_Tests/session_profiles/DUTSSH.ffsp
2.2	command	s1	time
2.3	close	s1	
	procedure		**GetDate**
1	lock		s1_lock
1.1	open	s1	project://Synchronization_Tests/session_profiles/DUTSSH.ffsp
1.2	command	s1	date
1.3	close	s1	

Timeout

If a step times out, iTest generates an **OnStepTimeout** event and execution continues at the first step after the locked steps.

signalWait: Sleep the currently executing thread

signalWait causes the currently executing thread to sleep until it is resumed by a call from **signal**, **signalAll**, or **signalActivate** with any one of the specified event names.

Action	**Command property value** (in the **Description** cell)
signalWait	*eventName* [, *eventName, ...*]

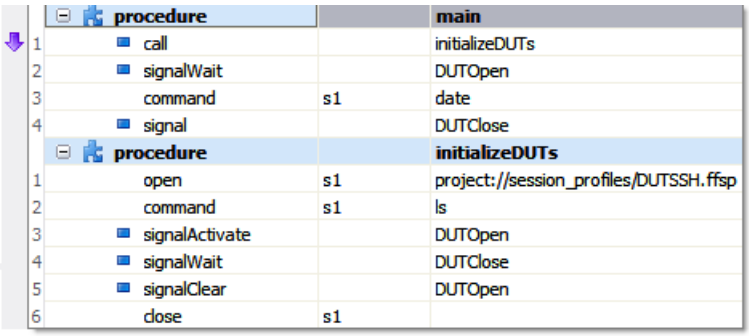
Note In contrast to Java's implementation, threads that are waiting do not release locks and threads do not need to be inside locked blocks to call a wait.

Example

a The **signalWait** in step 2 of the **main** procedure ensures that the **date** command in step 3 of **main** does not occur until the **DUTOpen** event is active.

b The **DUTOpen** event is activated by the **signalActivate** in step 3 of the **initializeDUTs** procedure.

c As a result, the **date** command can not occur until it is certain that the session with the DUT is open)



	procedure		main
1	call		initializeDUTs
2	signalWait		DUTOpen
3	command	s1	date
4	signal		DUTClose
	procedure		**initializeDUTs**
1	open	s1	project://session_profiles/DUTSSH.ffsp
2	command	s1	ls
3	signalActivate		DUTOpen
4	signalWait		DUTClose
5	signalClear		DUTOpen
6	close	s1	

Timeout

If a step times out, iTest generates an **OnStepTimeout** event and execution continues past the **signalWait** step.

signalWaitAll: Wait until all specified events have been signaled or activated

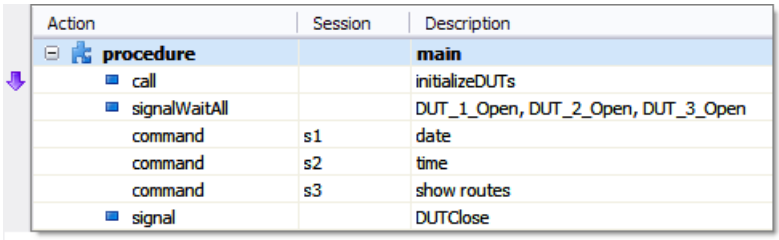
signalWaitAll causes the currently executing thread to sleep until all specified events have been signaled or activated (by **signal**, **signalAll**, or **signalActivate**).

Note In contrast to Java's implementation, threads that are waiting do not release locks and threads do not need to be inside locked blocks to call a wait.

Action	**Command property value** (in the **Description** cell)
signalWaitAll	*eventName* [, *eventName, ...*]

Example

In this example, the **signalWaitAll** step ensures that the three commands to three different sessions do not occur until all of the three events that signal the open status of the three sessions are activated.



Action	Session	Description
procedure		**main**
call		initializeDUTs
signalWaitAll		DUT_1_Open, DUT_2_Open, DUT_3_Open
command	s1	date
command	s2	time
command	s3	show routes
signal		DUTClose

Timeout

If a step times out, iTest generates an **OnStepTimeout** event and execution continues past the **signalWaitAll** step.

signal: Wake a thread that is waiting on an event

A **signal** *eventName* step wakes one thread that is waiting on *eventName* and causes it to continue execution.

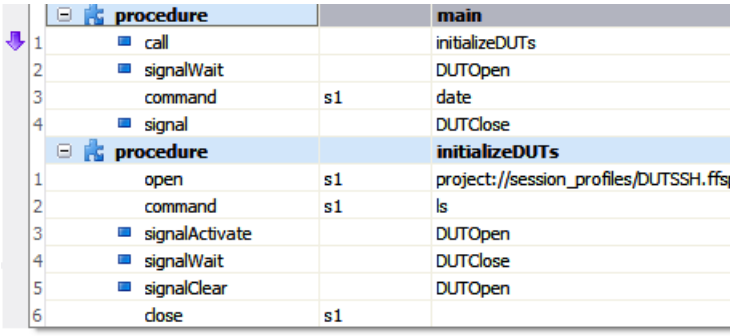
- If several threads are waiting on *eventName*, then one randomly selected thread continues and the rest continue to wait.
- If no threads are waiting on *eventName*, then *eventName* remains signaled until a thread consumes the event or a **signalClear** action removes it.

Action	**Command property value** (in the **Description** cell)
signal	*eventName*

In addition to configuring **signal** as an Action in a step, you can specify **signal** as an Action for an Event.

Example

The **signalWait** in step 4 of the **initializeDUTs** procedure can proceed only when the **DUTClose** event is signaled. In step 4 of the **main** procedure, the **signal** action signals the **DUTClose** event to enable the thread that is executing the **initializeDUTs** procedure to proceed.



signalAll: Wake all threads that are waiting on an event

If there are any threads waiting on *eventName*, then **signalAll** *eventName* wakes all of the threads and causes them to continue execution. If no threads are currently waiting on *eventName*, then the **signalAll** step does nothing.

Action	**Command property value** (in the **Description** cell)
signalAll	*eventName*

Tip If no threads are currently waiting on *eventName* and you want the event to "stay around" until explicitly "told not to", then use **signalActivate** instead.

In addition to configuring **signalAll** as an Action in a step, you can specify **signalAll** as an Action for an Event.

signalActivate: Turn a signal on

A **signalActivate** *eventName* step turns on the event called *eventName*. While an event signal is activated, any threads currently waiting for the event will be allowed to continue and any threads that begin waiting for the event are allowed to continue until the event is deactivated by a **signalClear** action..

Action	**Command property value** (in the **Description** cell)
signalActivate	*eventName*

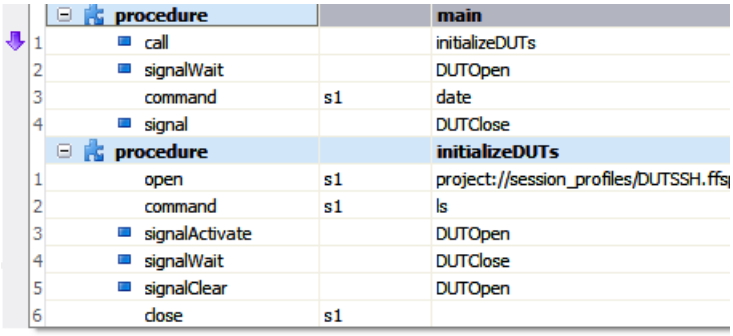
In addition to configuring **signalActivate** as an Action in a step, you can specify **signalActivate** as an Action for an Event.

Example

a The **signalWait** in step 2 of the **main** procedure ensures that the **date** command in step 3 of **main** does not occur until the **DUTOpen** event is active.

b The **DUTOpen** event is activated by the **signalActivate** in step 3 of the **initializeDUTs** procedure.

c As a result, the **date** command can not occur until it is certain that the session with the DUT is open)



signalClear: Deactivate a signal

A **signalClear** *eventName* step removes any instances of the event named *eventName* that had previously been activated either by **signalActivate** or by **signal**..

Action	**Command property value** (in the **Description** cell)
signalClear	*eventName*

In addition to configuring **signalClear** as an Action in a step, you can specify **signalClear** as an Action for an Event.

Example

The **signalWait** in step 4 of the **initializeDUTs** procedure can proceed as soon as the **DUTClose** signal is sent by step 4 of the **main** procedure. Now that the session will be closed in step 6, we ensure that the **DUTOpen** signal is deactivated by performing a **signalClear** in step 5.

This programming practice ensures that any step in some other procedure that depends upon the session with the DUT being open cannot incorrectly try to execute in a session that is closed.

procedure		initializeDUTs
open	s1	project://session_profiles/DUTSSH.ffsp
command	s1	ls
signalActivate		DUTOpen
signalWait		DUTClose
signalClear		DUTOpen
close	s1	

waitThread: Wait for steps to complete

Use a **waitThread** step to ensure that execution does not continue until specified threads finish execution. A **waitThread** step completes only when the last thread finishes (you specify which threads to wait for). Execution then continues with the next step.

For example, the setup portion of a test case involves configuring three devices (using a **call** step to each of three different setup procedures). To speed up overall execution by executing the procedures concurrently, you configure each **call** step to run in a separate thread. You would use a **waitThread** step after the **call**s to ensure that execution does not continue until all three procedures finish. In this example, the **waitThread** step completes only when the last device is configured. Execution then continues with the rest of the test case.

Creating a waitThread step

1. For each step that you want to wait for, set the following property values (in the **General** property group):
   - Specify that the step should execute asynchronously: Check the **Start this step (in a new thread) and proceed to the next step** box.
   - Specify a name for the thread in the **threadName** property.

     Follow the naming guidelines listed in “Naming variables and procedures” on page 141.

     The name need not be unique. If multiple threads share a name, then the **waitThread** step is activated only when the last thread with the shared name finishes.

     Because field replacements are supported in the text, you can define a name that can be generated dynamically (for example, in a loop with the loop count as the replacement text).
2. Create the step that will wait:
   - After the asynch steps, add a step with an EXEC action of **waitThread**.
   - Specify the threads that the **waitThread** step should wait for: In the **Command** property, specify the **threadName** values of the threads. This can be a wildcarded list and can make use of field substitution.

Conditions

- If iTest encounters a **waitThread** step and none of the currently active thread is one of the threads listed in the **waitThread** step's command, then execution continues immediately.
- If multiple threads have the same name, then the **waitThread** step completes only when the last thread with that name finishes.

killThread: Kill the specified threads

Use a **killThread** step to immediately stop execution of the specified thread. A **killThread** step completes when the thread finishes. Execution then continues with the next step.

Creating a killThread step

1. Add a step with an EXEC action of **killThread**.
2. Specify the threads that the **killThread** step should kill: In the **Command** property, specify the **threadName** values of the threads. This can be a wildcarded list and can make use of field substitution.

Conditions

- If iTest encounters a **killThread** step and none of the currently active threads are listed in the **killThread** step's command, then execution continues immediately.
- If multiple threads have the same name, then the **killThread** step completes only when the last thread with that name finishes.

CHAPTER 37

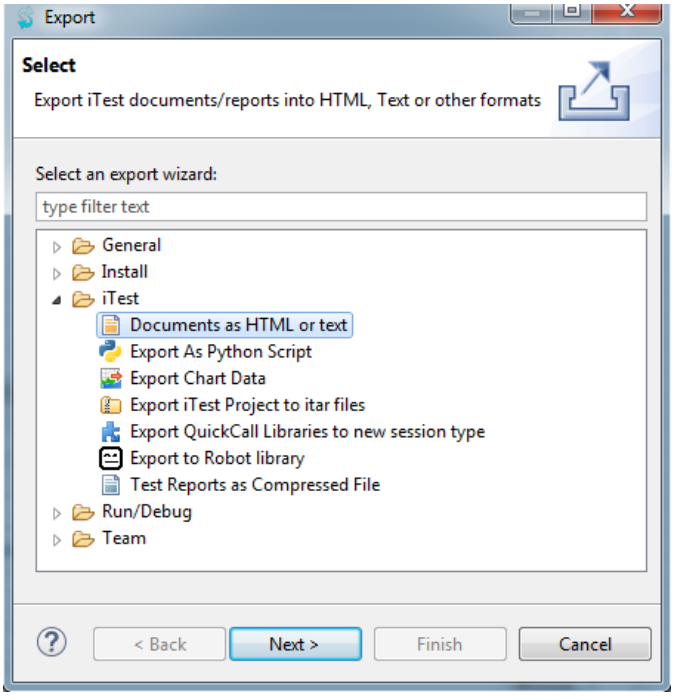
Sharing iTest Resources

Exporting test cases and other iTest documents

Use the procedure described in this topic for most iTest document types. For test reports, there's an easier way — see "Uses for test reports" on page 426.

Export document as HTML, XML, or text

1. In the iTest explorer, right-click the document and click **Export**. The Export wizard opens.
2. On the **Select** page, select **iTest > Documents as HTML or text**. Click **Next**.



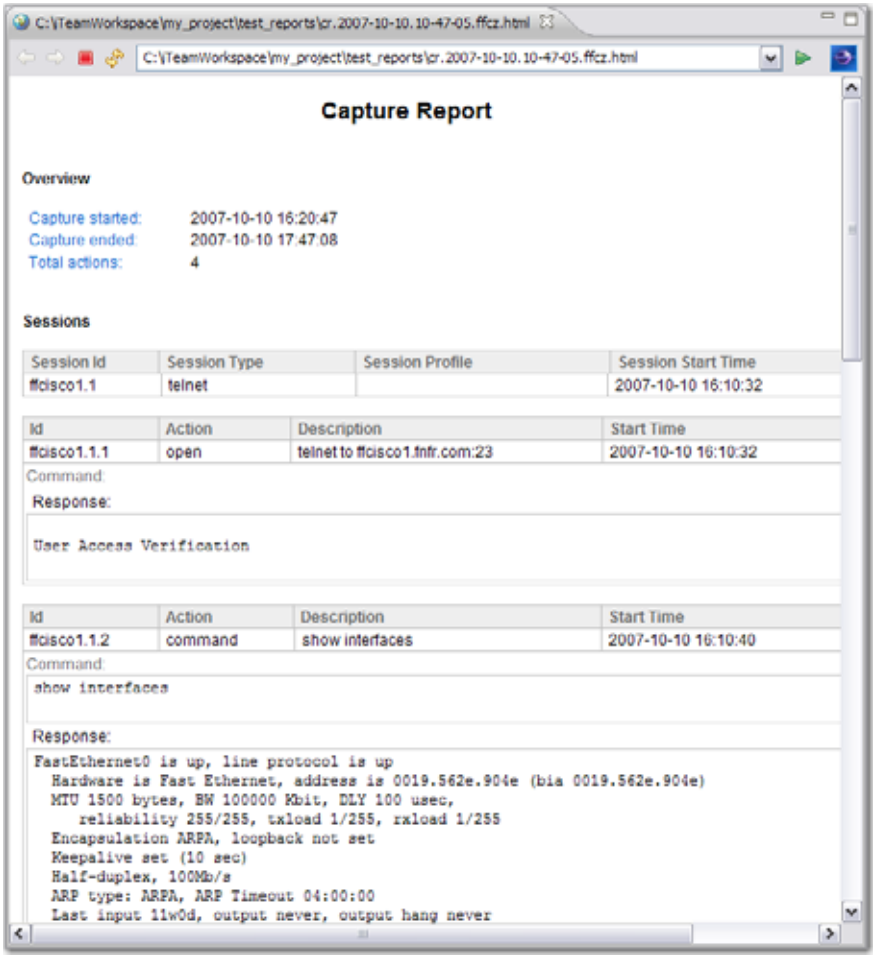
3. The **Export** page confirms the file to export. Click **Next**.
4. On the **Report Style** page, for **Stylesheet folder**, specify the output file format. Some file types offer additional options like **summary** and **detail** versions. Click **Next**.

5 On the **Report Output Location** page, the **File name** field confirms the filename with the appropriate filename extension. You can modify the output folder (**Location**) and **File name** as needed.

Optional: Check **View the generated report in editor** to view the resulting file in an editor in iTest.

Click **Next**.

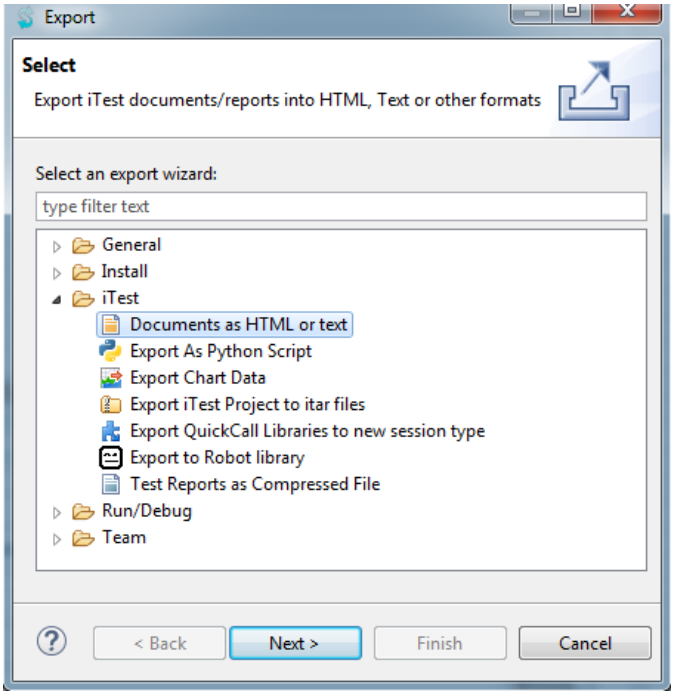
6 Click **Finish**. The file is added to the specified folder and, if specified, the file opens in an editor, as shown in this example.



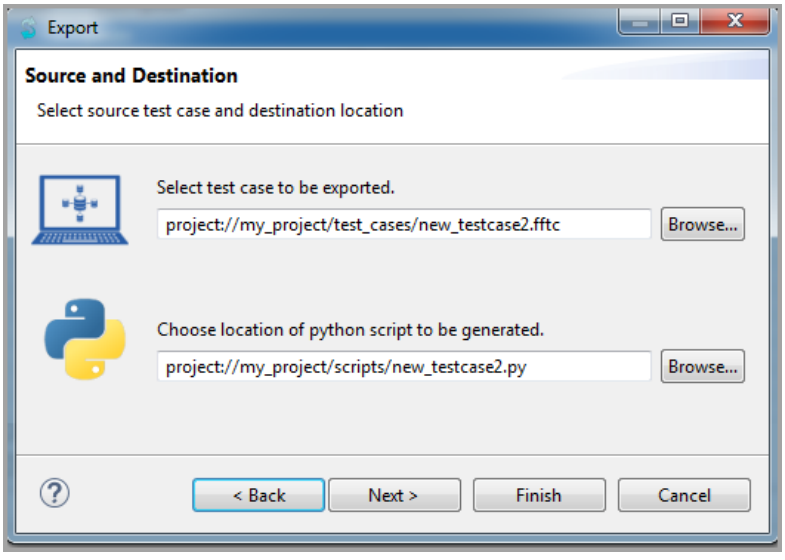
Export test case as Python Script

1 In the iTest explorer, right-click the document and click **Export**. The Export wizard opens.

2 On the **Select** page, select **iTest > Export as Python Script**. Click **Next**.



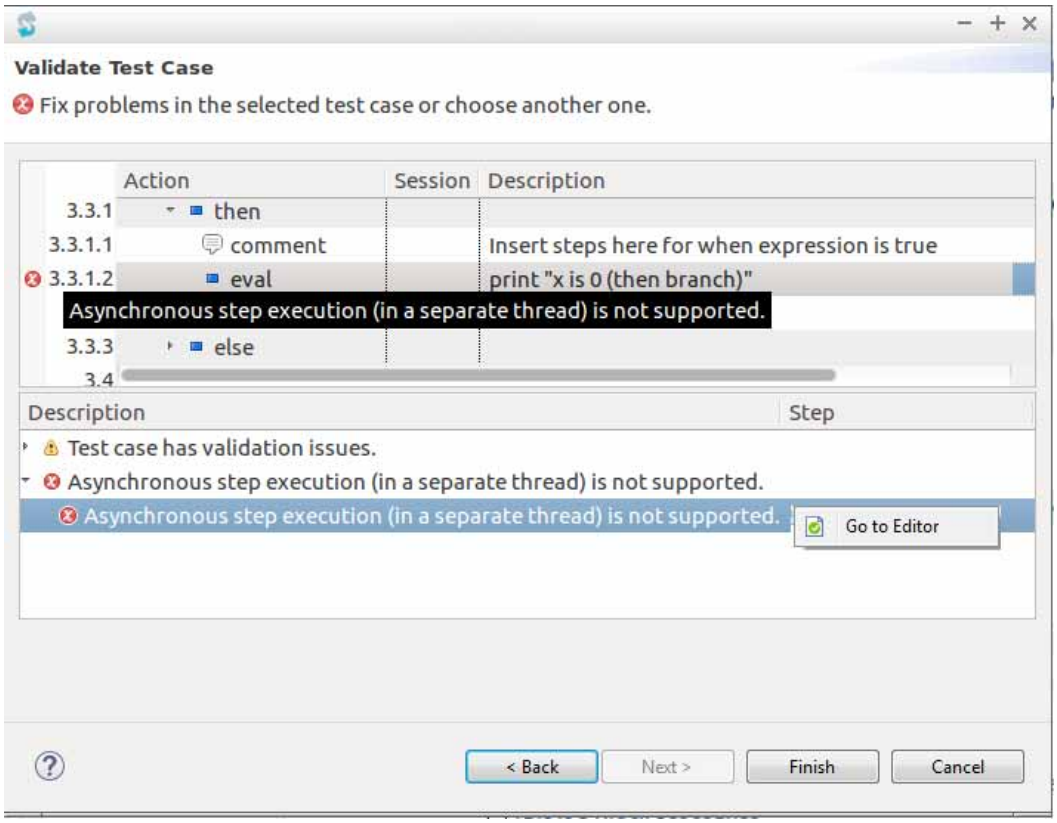
3 On the **Source and Destination** window, select source test case to convert and the location where the generate Python Script will be saved.



- **Select test case to be exported:** Click Browse. Default location is Workspace. Navigate your workspace or the file system and select the file to be exported as Python Script and click **OK**.

- **Chose location where Python script will be generated:** Click Browse. Default location is **my_project/scripts** and the default name is the **test case file name.py**. Browse to a location of your choise and click **OK**.

4 Click **Next** and the **Validate Test Case window** displays.



- The test case step view displays at the top of the window and a list of validation issue displays at the bottom.
- Each validation issue from the list displays a description and a step id (if any).
- Double-click on the validation issue at the bottom of the window and the corresponding step shown in steps view (if any) displays.
- Right-click this validation issue or a step, to display the **Go to Editor** menu.
- Click **Go to Editor** and the corresponding test case opens.

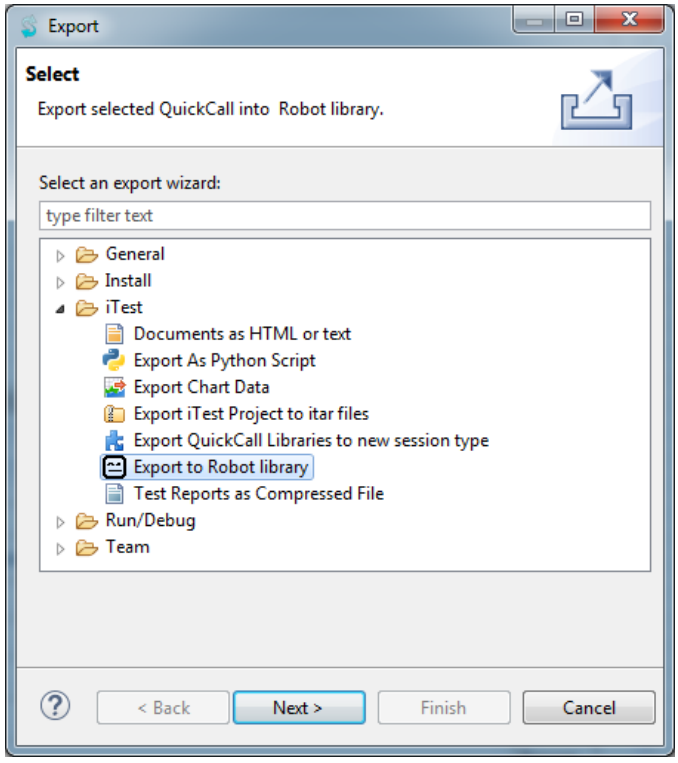
5 Click **Finish** to convert the selected test case to Python.

iTest converts test case to Python irrespective of any validation issues, which you may edit as required.

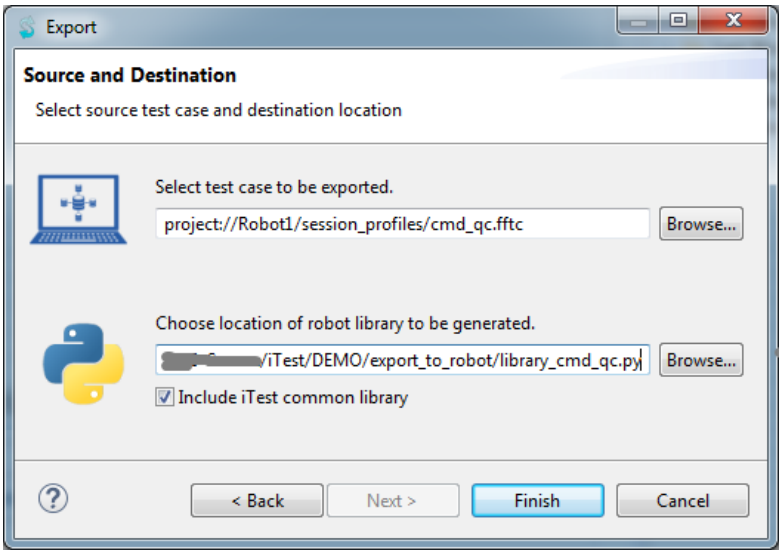
Export test case to Robot library

1 In the iTest explorer, right-click the document and click **Export**. The Export wizard opens.

2 On the **Select** page, select **iTest > Export to Robot library**. Click **Next**.



3 On the **Source and Destination** window, select source QuickCall to convert and the location where the generate Python Script will be saved.



- **Select test case to be exported:** Click Browse. Default location is Workspace. Navigate your workspace or the file system and select the file to be exported as Python Script and click **OK**.

  You may select only one QuickCall at a time and export to Robot Keyword file.

- **Chose location where robot library will be generated:** Click Browse. Default location is **file://C:/Users/user-01/** and the default name is the **QuickCall file name.py**. Browse to a location of your choice and click **OK**.

- **Include iTest common library:** Indicates whether the iTest common keyword file should be generated to the robot library file folder. This option is selected by default.

  When selected: The export wizard will include the iTest common keyword file—**iTestCommon.py**, in the robot library file folder.

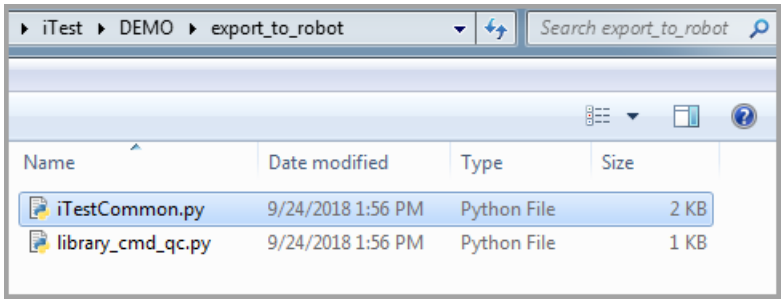
  This file is a support file required to use iTest exported Robot Keyword file in Robot script and execute in an external environment. You cannot execute QuickCall library without this file.

  When not selected: The export wizard will not generate the iTest common keyword file—**iTestCommon.py**.

4 Click **Finish** to convert the selected QuickCall to Robot framework file or click **Cancel** to discard the export operation.

iTest generates a Python file, example: **library_<quickcall>.py**, which a robot framework user can import it to robot test case.

If you had selected to **Include iTest common library**, iTest will generate an additional python file—**iTestcommon.py**. See the example below.



If the file exists, iTest displays a dialog asking you to confirm whether you wish to overwrite the existing file or change file name.

Sharing projects with colleagues and saving them for use in automated testing

This topic describes sharing iTest files, for example:

- Sharing a test case with a coworker that uses iTest (and easily including all supporting files in the package) so that they can run the test case under identical conditions.
- While every test case developer uses the “official” set of response maps, there is no need for each developer to have a copy of the files in their workspace. Instead, share the files by storing them in a central file.

- Saving the full set of test cases for a particular release and all supporting files to the regression system (under source control) to support headless execution by iTestRT. The tests do not have to be in a iTest workspace for iTestRT to run them.

iTest files are interdependent; test cases depend on topologies or testbeds, topologies and testbeds depend on session profiles, session profiles depend on reference session profiles, and so on. One file might depend on a file in another folder in its project or on a file in a different project altogether. This means that, to ensure that all dependencies are met when sharing a particular file, you will actually export one or more projects to the file system.

Sharing using itar files

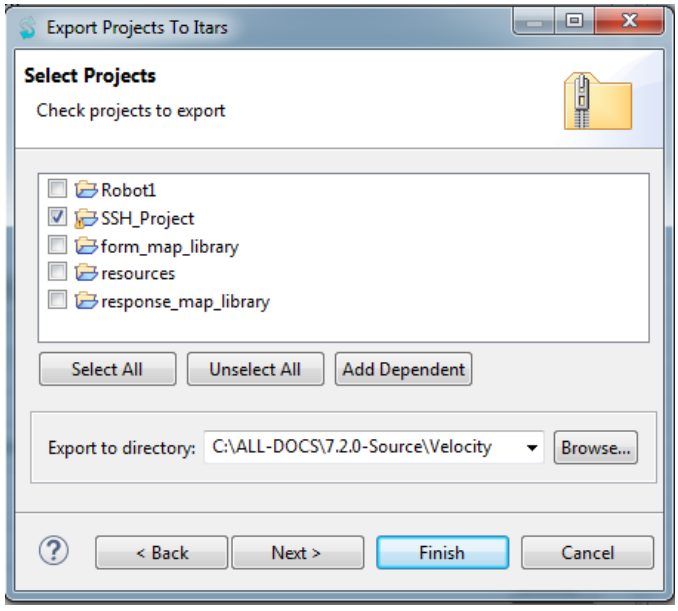
To share files, you export all related projects to ***itar*** files (**iT**est **a**sset **r**epository). An itar file enables any instance of iTest or iTestRT (regardless of workspace or location on the network), to use the included files. An itar file is a read-only zip file that contains all of the files that were contained in a single iTest project.

Exporting iTest projects as itar files

You export projects to an itar file in a folder that you specify.

1. First ensure that all projects are properly built. The easiest way to ensure a current build is to click **Project > Clean > Clean all projects**.
2. In the iTest Explorer, right-click the project and then select **Export**.
3. On the **Select** page, select **iTest > Export iTest Projects to itar files** and then click **Next**.
4. On the **Select Projects** page, select the project or projects to export. (If you started the wizard by right-clicking a project, then the project is selected for you.)

Tip To ensure that all referenced and dependent files are included, click **Add Dependent**. iTest exports each required project to an individual itar file in the folder specified in **Export to directory**. (Projects that are already stored in itar format are not “re-itarred”.)



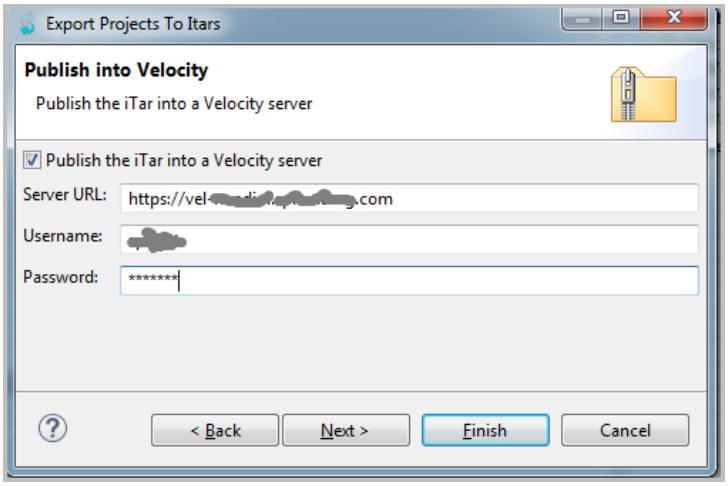
5 In the **Export to directory** field, specify the folder to export the itar files to.

For example, this can be a location in your regression system under source control. (For instructions on accessing files that are stored in itar files, see "Accessing iTest files that are held in itar files" on page 817.)

You have the following options:

- Save all itar files to a central location (typically under source control). Any reference to a file using a **project://** URI in an instance of iTest or iTestRT will look in this location to find files that are included in an itar file.
- While browsing to the folder, create a subfolder directly under a shared workspace root directory and name the subdirectory **itar**. Any instance of iTest will, by default, look in this location to find files that are included in an itar file.

6 Click **Next** and the **Publish into Velocity** page opens.



Publish the iTar into a Velocity Server	Not selected by default. When **not selected** the iTar files will be created in the location specified in Step 5. Select the option to upload iTar files to the Velocity server.
Server URL **Username** **Password**	These options become available when you select **Publish the iTar into a Velocity Server.** It is mandatory to enter correct Velocity **Server URL**, **username**, and **password**. An error message displays if any of the information you entered is invalid or is missing. **Note** The **Server URL**, **username**, and **password** will be populated only if you have set up these details in **iTest > Windows > Preferences > Spirent > Velocity.**

7 Click **Finish** to create the itar files. Each itar file is named with the name of the associated project.

Publish the iTar into a Velocity Server	The iTar file are created, and automatically published in the Velocity server. A progress bar shows the task progression and displays the name of the file being published to iTAR before publishing to Velocity Server.
Export to directory	If you are exporting to a directory (Step 5), and If there are existing iTar files with the same filenames as the files you are creating, you may choose to **Overwrite existing files** to replace the old files with the new ones. A progress bar shows the task progression and displays a message saying the projects are being exported to iTar and the name of the iTar file. In this example, we created the **R5000_router_response_maps.itar** file in the **C:\R5000Regression\ folder.** (The itar files for all checked projects will also reside in the **C:\R5000Regression\ folder**.)

Note If there is any error during publishing, iTest does the following:

- Indicates the files that caused the problem, and displays the associated error message from Velocity server for each of the files.
- Displays a message asking whether you want to continue and complete the publishing process and skip the problem files, or cancel the process and remove the processed files (on Velocity server).

iTest processes files according to your response.

Accessing iTest files that are held in itar files

iTest and iTestRT can access any file that is part of an itar file.

How iTest accesses files that are held in itar files

If iTest looks for a file in the current workspace and does not find it, then iTest.exe will look in following places for files that are held in itar files, in the following order:

- In any directory (or file) that is referenced using an `--itar` command-line option when eclipse/itest is started
- In a **itar** directory immediately under the workspace root
- In any directory (or file) referenced using an **ITAR_PATH** environment variable.

  To enable this capability, define an **ITAR_PATH** environment variable for the path. The **ITAR_PATH** environment variable uses the same conventions as the **CLASSPATH** environment variable (delimiters, quotes for paths that include spaces, and so on).

Note The path that you specify for **ITAR_PATH** cannot contain the path separator character.
Linux: Paths cannot contain ":"
Windows: Paths cannot contain ";"

How iTestRT accesses files that are held in itar files

Because iTestRT operates in the file system and does not work within a workspace, it looks for itar files in the following order:

- In any directory referenced using an `--itar` command-line option for iTestRT
- In any directory referenced using an **ITAR_PATH** environment variable (same as for iTest)

Viewing iTest files that are held in itar files

While using iTest, to view files and folders that are held in itar files, use the External Projects view, as described in "External Projects view" on page 818.

Executing test cases that are held in itar files

iTest looks for files in itar files in the following order:

- If a **run** step refers to the URI **project://com.fnfr.project2/icmp_echo_verify.fftc**, then iTest will look for a project named **com.fnfr.project2** in the workspace.
  - If found, then it will look in the project for **icmp_echo_verify.fftc**.
  - Otherwise, it will look in the **<workspaceName>/itar** directory for a file named **com.fnfr.project2.itar**. It will look in the itar for **icmp_echo_verify.fftc**. If found, it will be used.
  - Otherwise, it will look in all the directories in **ITAR_PATH** for a file named **com.fnfr.project2.itar**. It will look in the itar for **icmp_echo_verify.fftc**

Note Once iTest finds a matching project source, no additional sources will be searched, even if that first source does not contain the path or file requested.

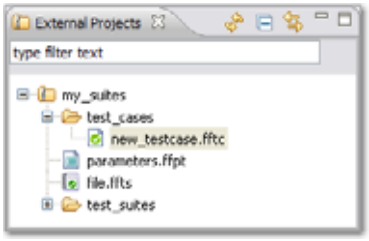
External Projects view

For the directories that appear in the in the **<workspaceName>/itar** directory or that are specified by the **ITAR_PATH** environment variable, the External Projects view displays all projects, folders, and files that are held in itar files. Each itar file is shown as a iTest project (the associated icon includes a zipper to indicate that it is "zipped").

The External Projects view looks to the two locations in the following order. If there are multiple projects with the same name in multiple locations, then the External Projects view displays only the first project that it finds.

- The **<workspaceName>/itar** directory
- The directories specified by the **ITAR_PATH** environment variable

Note If there is a project with the same name in both the current iTest workspace and in any location that iTest searches for itar files, then the project will appear in the iTest Explorer only and not in the External Projects view. That is, any project name will appear in only one view in the iTest window.



CAUTION Keep in mind that there can be some confusing interactions when, in addition to the projects in your workspace, you make use of projects in itar files. If you export a project to an itar and then use the iTest Explorer to delete the file, it is removed from the workspace but not from the file system. As a result, the itar'ed project might now appear on the External Projects view. The next time that you start iTest, however, iTest auto-imports the original project (because iTest discovered it in the workspace). The project now appears in the iTest Explorer once again. To use only the itar'ed version, you must use the operating system file management utility to remove the project from the workspace.

- To open the External Projects view, click Show View in the main toolbar and then select **External Projects**.
- Notice that the familiar "hanging folder" icon for projects does not appear because the view displays the contents of the itar files in the computer's file system and not the contents of a workspace. The "zip folder" icon represents itar files.
- Click **Collapse All** to collapse the directory tree for easier navigation.
- To view particular files, type a search string into the filter text box at the top. You can use * and ? wildcard characters. Only files with matching text appear in the view. Click **Clear** to remove the filter text.
- To execute a test case, select it and then click **Start Execution in New Window** in the main toolbar. By default, when you start a session using a session profile from the Favorites view, the iTest Explorer view, the External Projects view, or the Session Profile editor, the session starts in a new session window.
- Double-click a file to open it in the appropriate editor. Alternatively, right-click it and select **Open**. Remember that any file in an itar file is read-only — you will not be able to modify the file. You can, however, make changes and then save the file with a different name.
- To copy a files URI into the clipboard, right-click the file and select **Copy URI**
- In any folder, right-click a document to view a menu of options.

External Projects view toolbar

	Refresh the view to display recent changes. Alternatively, press F5.
	Collapse all open folders so that only the top-level folder appears in the view.

CHAPTER 38

Testing High-Availability (HA) Devices

Testing HA devices: Overview

HA devices are special in that any number of redundant processors (nodes) can perform system operations (that is, act as the *master*). Without special features, HA test cases would be quite complicated:

- The test case would have to start a session with each node.
- You would have to create logic that somehow determined which node is currently master, and then the test case would send the appropriate commands to session with the master.
- If mastership moves to another node, the test case would have to somehow detect the hand-off and redirect the commands appropriately.

Happily, you can skip all this because iTest enables you to test HA devices in a simple, intuitive way.

Important iTest does not capture HA sessions.

iTest HA Operation

- iTest's HA feature treats the HA device (with multiple redundant nodes) as a single virtual device — iTest takes care of directing commands to the appropriate node.
- HA supports multiple connections inside a single session (either Telnet or SSH) connected to nodes via Telnet/SSH sockets to different IP addresses and/or port numbers. One HA session window appears for each connection.
- By default, iTest determines master/slave/other state based on the prompts returned to commands (you specify the master/slave/other prompts in the session profile or testbed device).
- In the most common situations, test cases send commands to the master. You can override the session's default HA behavior by setting a property for any step to direct commands to a slave node or to a specified node.
- Responses from the master node appear in the Response view and responses from other nodes appear in the structured data in the Structure view
- You can configure what should happen when an intended recipient (be it master, slave, or other) cannot be found, including an option to poll waiting for a recipient to be identified.
- You can use **setmaster** and **setslave** actions to explicitly set master/slave status.

Testing HA devices

Step 1: You prepare to test an HA device by setting a few HA properties for the SSH or Telnet session. In the Testbed editor or the Session Profile editor, enable HA operation by configuring the properties of the HA device.

Step 2: Specify the prompt that you expect the master node to return that identifies it as the master. Do the same for the slave (and "other") nodes.

Step 3: Now create the test case.

- A single **open** step opens connections with all nodes. If needed, you can add a login step for each node. For additional portability, you can parameterize the credential values (typically, username/password).
- By default, commands are directed to the master. If needed, you can direct particular commands to particular nodes.
- If needed, you can override any HA property setting for any step.
- A single **close** step closes connections with all nodes.

Testing HA devices: Detailed instructions

Step 1: Enable HA operation by configuring HA properties

Tip We recommend that you configure all HA property settings in the testbed document to ensure that the settings are portable and maintainable across topologies or testbeds and test cases.

---

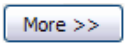
1 Specify the SSH or Telnet session type with HA nodes in either of the following locations:

- On the Testbed editor **Devices** page
- On the Session Profile editor **Start** page

2 In the **Telnet** or **SSH** property group, specify the **IP address** and **Port** values for the master node. (You can specify a hostname instead of an IP address.)

Note The IP address and port values that you just added identify the node with index value 0. In a later step, you will specify the IP address and port values for nodes 1, 2, 3, ... n.

---

3 If you are working in the Session Profile editor, click 

to open the **Session Properties** pages (the structured list of properties for a testbed device, session, or step).

4 In the list, select **Telnet (or SSH) > High Availability** to open the **High Availability** property group and set the following HA properties:

High Availability	Check the box to enable HA operation. (The default setting, unchecked, specifies normal, non-HA operation.)
Additional connections	Specify the IP address and port pair for each redundant node (nodes other than the master node.). This information is used only by the **open** step for a session. The values in the list represent nodes 1, 2, 3, ... n. Use the following format, one node per line: **<IP_or_hostname>:<portnumber>** **Important**: Be sure not to enter the values for node 0 — the master node — those values are specified by the **IP Address** and **Port** properties.

5 In the list, select **Terminal > Replay > Step Defaults > High Availability** to open the **High Availability** property group and set the following HA properties:

Verify status	If checked, then, during execution, before a **command** or **getstate** step (but not a **break** step), send an empty command to all nodes and then analyze the prompts to determine mastership. This setting ensures that the step sends the command to the correct node (in the case that mastership changed since the preceding step). The extra empty commands are not included in test reports. If **un**checked, then use the most recent prompt from each node to determine which is master. Default: checked
Wait for master / slave status	This setting specifies the action to take for **command** steps if the intended recipient of the command (specified by the **Send to** property) does not respond. If the box is not checked (default) and the intended recipient is not found, then iTest generates an **OnProcessorNotFound** event. If the box is checked and the **Verify status** box is checked, then: Every several seconds, send an empty command to each node and then assess the prompts to determine whether the intended recipient is available. Once the intended recipient is available, continue execution by executing the **command** step. The extra empty commands are not included in the Execution view or in test reports. Default: unchecked

Send to	Identifies the node to which a command will be sent for **command** and **break** steps. **Master**: (default) Send the command to the master — node 0 (based on the prompt or as set by a preceding **setmaster** command). **Slave**: Send the command to the first node (in index order) that is not Master and not Other. **Specific**: This setting is typically used to log in to a particular node and to test HA redundancy operation. Specify the particular node to which the command should be sent using the **Send to index** property.
Send to index	This property is used only if you set the **Send to** property to **Specific**. Specify the processor index (0, 1, 2, 3, ... n) to send the command to. node 0 is the node specified by the **IP Address** and **Port** properties. Nodes 1, 2, 3, ... n are specified in the **Other IP addresses** property.

Step 2: Define the HA prompts

During default HA operation, iTest determines mastership automatically by sending empty commands and using prompts to determine which node is master, which are slave, and which are "other". (The empty steps do not appear in the Execution view or in test reports.)

you will now define the prompts that identify the various HA nodes.

1 Open the **Prompts** page. Click **Terminal > Prompts**.

2 To enable you to add prompt definitions, check **Include additional values from list**.

3 Click  and then specify the prompt properties as usual. (See "Editing prompt definitions" on page 467.)

4 For the **High Availability indication** property, specify the type of node that you are defining the prompt for. (The default value, **Normal**, indicates a normal, non-HA prompt.)

The setting determines what the prompt indicates about the node that returned it:

- **Master**: The node that returned the prompt is an HA master node
- **Slave**: The node that returned the prompt is an HA slave node
- **Other**: The node that returned the prompt is neither master nor slave

---

Note Typically, mastership is determined after login completes for all nodes. For this reason, you should associate **Master/Slave/Other** status with prompts that do not appear as login or password prompts.

During execution, when a step is attempting to determine the master/slave/other state of the nodes, iTest compares the returned prompt to each configured prompt in order. The state is assigned based on the first match.

- If there is a match with a prompt for which **High Availability indication=Master**, then the node is considered a master.
- If there is a match with a prompt for which **High Availability indication=Slave**, then the node is considered a slave.
- If there is no match, then the node is considered **Other** (neither a master nor slave).

Step 3: Create the test case

iTest does not capture HA sessions, so you create test cases by manually adding steps in the Test Case editor.

The process of creating an HA test case is nearly identical to the normal process for non-HA devices. Once you configure the properties as described in "Step 2: Define the HA prompts", iTest sends all commands to the specified device (the master by default). For any step that should submit its command to another node, you specify the node in a property for the step.

Follow this procedure:

1 In the Test Case editor, add an **open** step that refers to the HA testbed device or session profile in the **Description** cell.

Note The **open** action for an HA session differs from the normal **open** action — it opens a connection with each HA node that is specified in the testbed device or session profile. As a result, you need only one **open** step to connect to all nodes.
iTest keeps all connections open as long as they stay open or until a **close** step is executed for the HA session. If one of the connections is closed by the server during the test, then subsequent steps operate only on the remaining connections.

---

2 If you must log in to each node to begin testing:

- **SSH**: The login process happens automatically. (For this reason, it is easier to implement SSH than Telnet.)
- **Telnet**: you will send the login commands to each node separately. See "Logging in separately to each HA node" on page 826 for instructions.

3 Add steps as needed. See "HA command reference" on page 826. By default, all commands are set to the master. If needed, see "Sending a command to a particular HA node" on page 827.

4 Add a single **close** step to close all connections.

Note The **close** action for an HA session also differs from the normal **close** action — it closes all connections to nodes for the HA session.

HA command reference

The following commands are available for any SSH or Telnet HA session.

For any step, you can override the default HA behavior for the testbed device or session property by setting the **Send to** property to direct the command to master, slave, or to a specified node.

setmaster

The **setmaster** action specifies the node to which commands will be sent for steps where the **Send to** property is set to **Master**. Commands with a **Send to** property value of **Master** are sent to the first node that is master.

For **setmaster** steps, specify an index in the **Description** cell to indicate the node to set as master.

setslave

The **setslave** action specifies the node to which commands will be sent for steps where **Send to** property value is **Slave**. Commands with a **Send to** property value of **Slave** are sent to the first node that is slave.

For **setslave** steps, specify an index in the **Description** cell to indicate the node to set as slave.

getstate

The **getstate** action returns:

- The current state of the processors in structured data (for querying and analysis) (as displayed in the Structure view)
- A human-readable table in the response (as displayed in the Response view)

Logging in separately to each HA node

If the device requires you to log in separately to each node, then you will define a set of steps for each node by setting the Send To property for the steps appropriately.

- **To log in to each node separately (SSH):**

  You do not need to create login steps because iTest logs in for you using the SSH credentials that you specified in the testbed document.

- **To log in to each node separately (Telnet to terminal server):**

  You do not need to create login steps because the terminal server is typically logged in permanently.

- **To log in to each node separately (Telnet to the device):**

  1. Create a normal login sequence for each node (typically one **command** step that sends the login ID and a second **command** step that sends the password).
  2. For each step, set the **Send to** property to **Specific** and set the **Send to index** property to the node's index value. See "Sending a command to a particular HA node" on page 827.

Tip Once you have developed the steps that log in, use them as the basis for a login procedure that any test case can call. The procedure should include a single **open** step and then steps that log in to each node in succession.

Sending a command to a particular HA node

Follow this procedure to create a **setmaster** or **setslave** step (described in "Specifying that a particular node should be master (or slave)" on page 828) or to send a **command** or **break** to a particular node (for example, when logging in):

1 Specify a value for the **Send to** property:

- **Master**: (default) Send the command to the master node (based on the prompt or as set by a preceding **setmaster** step).
- **Slave**: Send the command to the first node (in index order) that is not Master and not Other.
- **Specific**: This setting is typically used to log in to a particular node and to test HA redundancy operation. Specify the particular node to which the command should be sent using the **Send to index** property.

2 Specify the node by setting a value for the **Send to index** property: Type the index value of the intended recipient of the command. See "Specifying a node by index" on page 828 for instructions on determining the appropriate index.

Ensuring that a command is sent to the intended recipient

For **command** or **getstate** steps that are intended for a particular node (as specified by the **Send to** property value), you can specify that, before the command is sent, iTest first definitely determines the master/slave/other states of the nodes by checking the **Verify status** property.

iTest achieves this by sending an empty command that causes each node to return a prompt, thus indicating its state. As a result, you can be sure that the command will be sent to the recipient that you specified using the **Send to** property.

- This feature does not apply to **break** steps.
- Normal step **Timing** property settings apply for the empty commands.
- The empty commands do not appear in the Execution view or in test reports.
- The structured data includes the state of each node and the most recent prompt returned by the node.
- The **Prompt** data element contains the prompt returned in response to the actual command sent to the appropriate node.
- If there are multiple valid recipients (for example the **Send to** property value is **Slave** and multiple nodes return slave prompts) then the command is sent to the node with the lowest index value.

To ensure that a command is sent to the appropriate node:

1 Select the HA step (the action is one of the following: **command**, **break**, **getstate**, **setmaster**, or **setslave**).

2 In the **Step Properties** section, select the **Telnet <action name> Properties** node in the tree (for example, **Telnet setmaster Properties)**.

3 On the **HighAvailability** page, check the **Verify status** checkbox.

Tip If **Verify status** is not checked, then the command is sent to the last node that returned a master prompt. You can use the **setmaster** action to explicitly set the master (until the next **setmaster** or the next step where **Verify status** is checked).

Specifying a node by index

When you create a **setmaster** or **setslave** step or when you need to send a command to a particular node (for example, when logging in), you specify the node by index value. Nodes are assigned index values in the order in which you define them in the testbed device or session profile.

Specifying that a particular node should be master (or slave)

During default HA operation, iTest determines mastership automatically by sending empty commands and using prompts to determine which is master. In the case that you cannot distinguish master/slave using the prompts, you might want to specify that a particular node should be master.

you will use the **setmaster** action to explicitly set a particular node to master (and **setslave** to set a slave). As a result, the master node becomes the intended recipient for all steps for which the **Send to** property is set to **Master**. Steps with the **Send to** property set to **Slave** are sent to the first node (in index order) that is not master.

◆ To specify which node should be master (or slave)

1 Create a step and select **setmaster** (or **setslave**) in the Action cell.

2 In the **Description** cell, type the index number that identifies the node that you want to set to master (or to slave). See “Specifying a node by index” on page 828.

Viewing the current states of all nodes

A getstate step returns an XML table with the current master/slave and index number states of all nodes.

The return data depends on the setting of the **Verify status** property.

- If **Verify status** is checked (default), then iTest refreshes state information by sending state verification commands to the nodes before it polls for responses to the **getstate** step.
- If **Verify status** is unchecked for the step, then **getstate** returns the current states.

◆ To set the ‘Verify status’ property

1 Select the **getstate** step.

2 In the **Step Properties** section, select the **Telnet getstate Properties** node in the tree.

3 The **Verify status** checkbox appears on the **HighAvailability** page.

CHAPTER 39

Charting Test Case Data

Charting test case data

You can generate charts of the data returned during test execution. Charts are updated in real time and appear in the Charts view (**Window > Show View >Charting**).

Note You can chart only numeric data.

To generate a chart, you associate an analysis rule with the step. The rule generates the data to be charted. Analysis rules have two main parts:

- The **extractor** (the **Extract using** cell) that defines how to extract the value from a response (using regex or response maps, for example)
- The **processor** (the **Perform** cell) that processes the data. You'll use the **chart** processor to generate charts.

You can generate the following kinds of charts:

- X-Y graphs: area, line, scatter, or time series
- Bar charts
- Pie charts

Overview of the process of creating and viewing a chart

To generate a chart, you follow this overall process:

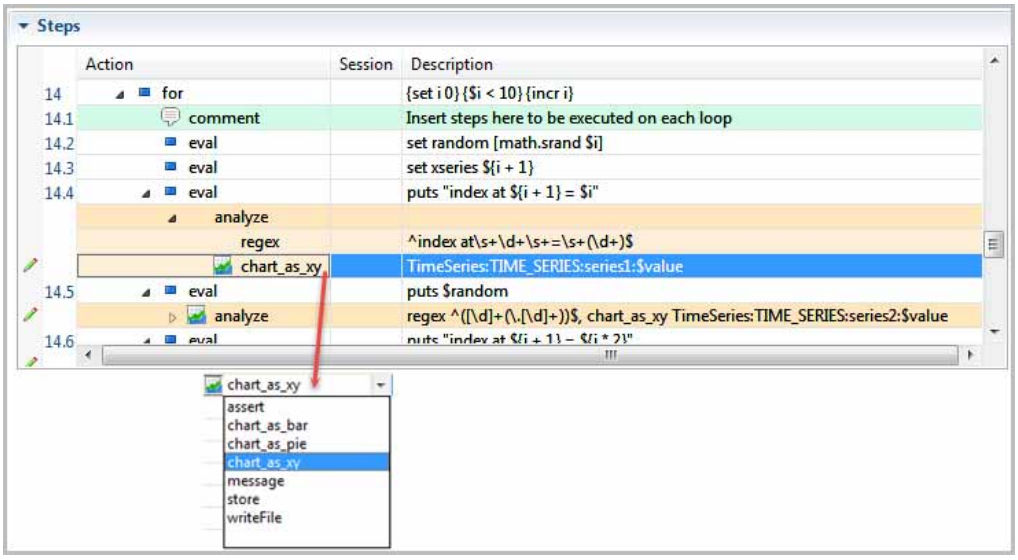
1. **Set Charting View Preferences.** Go to "Setting preferences for Charting View" on page 842 and select the chart color.
2. **Decide what you want to chart** and therefore which values to extract. To view a time-series, you'll need to repeat the step that generates the value (typically by placing it within a loop). To view one value as a function of another, you'll need to extract one value, store it using an analysis rule, and then extract the other value. You can use any type of extractor to extract values; Regex, queries based on a response map,
3. **Execute the test case** so that iTest has an example response for the step.
4. **Create an analysis rule.** In the Response view, select the value to chart and then right-click to create an analysis rule that extracts the value and applies the appropriate chart processor (X-Y, bar, pie).
5. **Specify property settings** for the chart.
6. **Open the Charts view.**

7 **Execute the test case.** Chart values are updated in real time and the chart remains visible when execution stops. If you chart a value for a long-running test, you can specify how much of the most recent data the chart should display.

Example: Charting a value as a function of time

Let's work through an example of charting a value as it changes over time (that is, charting the value as a function of time — a time series).

Our example test case opens a session with a device and then submits a command that requests IP traffic values. The step is contained within a **for** loop that repeats the command ten times.

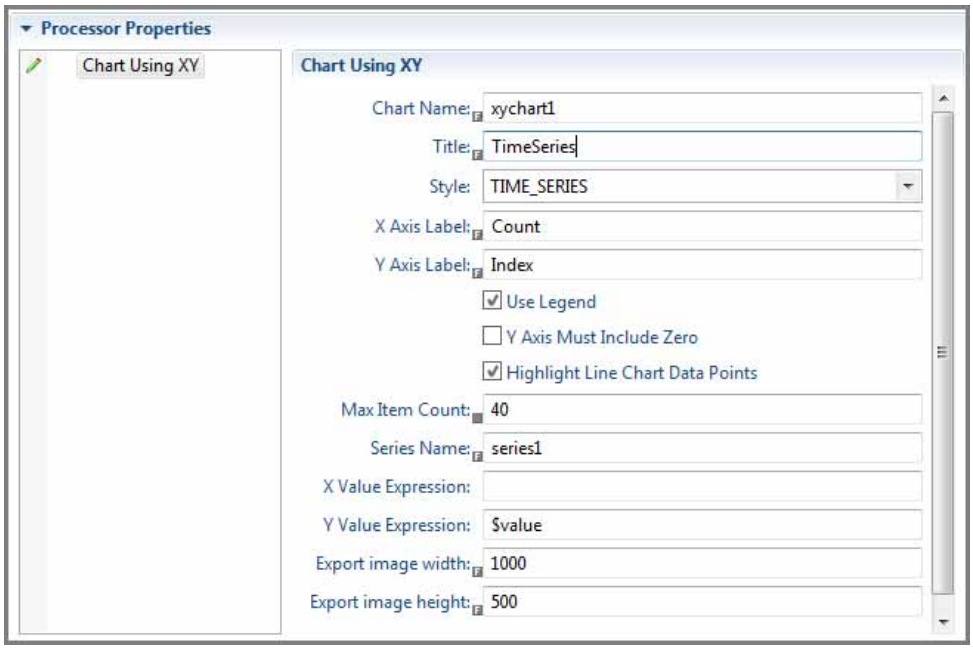


As a result, we might expect that the device should respond with ten different values, one for each time that the step executes.

You may also select the step so we can view the most recent response in the Response view, select the value to chart, right-click the value to add an analysis rule that extracts the value and charts it (**Quick Analysis Rule > Regulae expression>**).

Note iTest adds the analysis rule and the **Perform** action charts the value.

Select the Step Chart_as_xy, click open the **Procesor Properties** section. Click the **Chart Using XY** property group.



The **Chart name** property is iTest's identifier for this chart — analysis rules specify a particular **Chart name** to add data to a particular chart. This is helpful when several variables appear on a single chart or when you will chart multiple values on independent charts during execution.

- Enter a **Title** for the graph and label the **X axis** and **Y axis**.
- Select **Highlight Line Chart Data Points** to mark each data point (round dots) on line charts. **Highlight Line Chart Data Points** is not selected by default.

Note When iTest extracts a value using an analysis rule, it puts the value into a variable named **value**. You can also set the extractor to **none** and use any expression or variable to extract data.

The **Y value expression** property value is **$value** — the extracted value for received packets. The **X value expression** property is blank. If you do not specify a value for the value along the **X axis** (the independent variable), then iTest plots time on the **X axis.**

Open the **Charts view** and then save and execute the test case. The data is charted during execution and remains in view when execution stops.

Viewing charts in reports

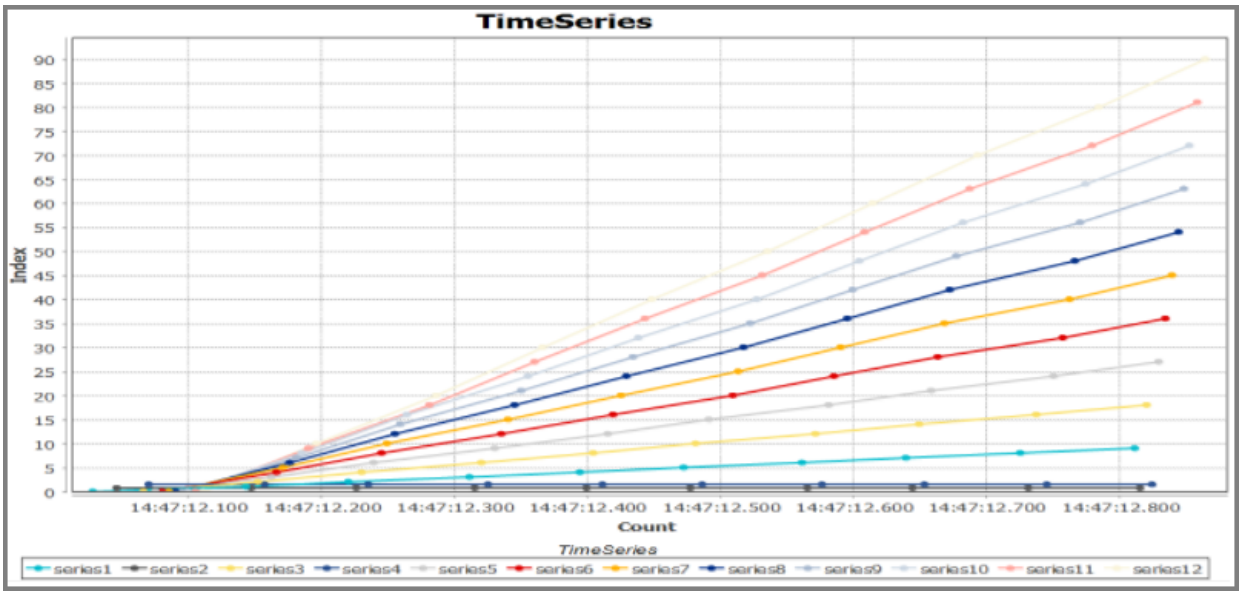
iTest displays charts in the Images view.

Charts are exported as JPEG files when you export a test report into HTML.

Charts view

The Charts view displays one or more charts of extracted data. The view opens whenever a test execution generates a chart. Charts are updated in real-time as the test case executes and remain visible when execution stops. You can design charts with the following formats:

- X-Y area
- X-Y line
- X-Y scatter
- X-Y time series
- Bar chart
- Pie chart



Charts view toolbar

	Display the previous chart (dimmed if no charts or only one chart are active)
	Display the next chart (dimmed if no charts or only one chart are active)

Specifying the appearance of charts: Chart properties

Once you have created an analysis rule that generates a chart (as described in "Charting test case data" on page 829), you specify the appearance of the chart as follows:

1. In the **Steps** section, select **Chart_as_xy**.
2. Click **Processor Properties** to expand and view, .
3. Select the chart type, **Chart Using XY** and view the the **Chart Using XY** window where you set the properties that control the appearance of the chart.

Details for each chart type follow.

Processors > Chart Using XY

The properties that you set in this section are represented in summary form in the **Details** cell of the analysis rule. You can edit the settings in either place.

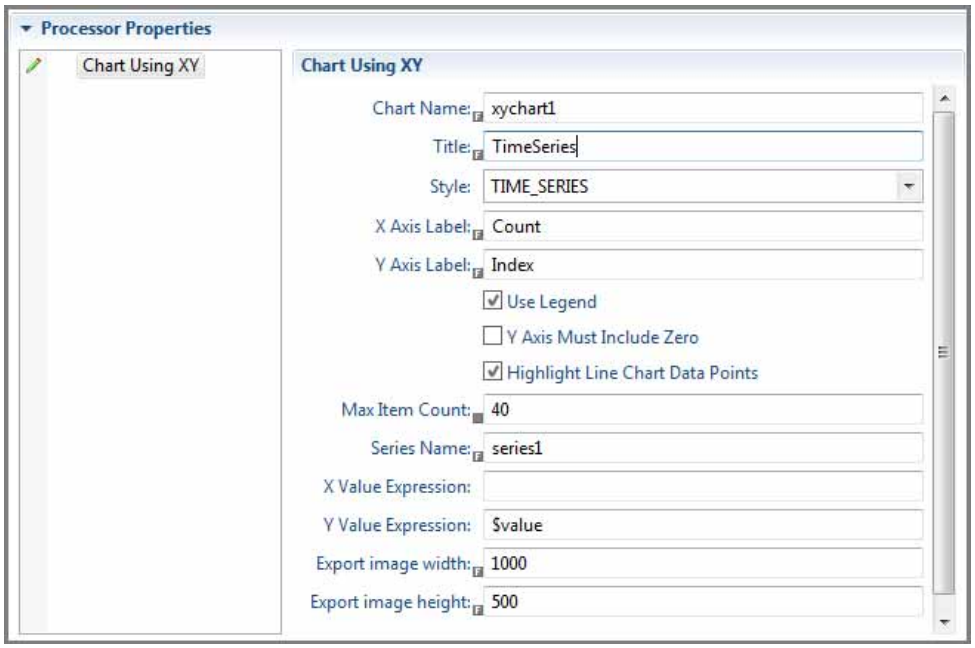


Chart name	iTest's identifier for this chart — analysis rules use the **Chart name** to add data to this particular chart. The label is ***not*** the Title of the chart. Instead, analysis rules from several steps and/or several analysis rules from a single step can send data to a particular chart by specifying the same **Chart name**. If only a single value is charted on the chart, then **Chart name** is not used.
Title	The text title that should appear at the top of the chart when you print or view it.
Style	Select the charting style that will be used to display: **AREA:** Area Graphs are Line Graphs with the area below the line filled in with a certain colour or texture. Area Graphs are drawn by first plotting data points, joining a line between the points, and then filling in the space below the completed line. **LINE:** A line chart displays information as a series of data points called 'markers' connected by straight line segments. **SCATTER:**Scatter plots are similar to line graphs in that they use horizontal and vertical axes to plot data points. However, Scatter plots show how much one variable is affected by another. The relationship between two variables is called their correlation . **TIME-SERIES:** A time series chart, is an illustration of data points at successive time intervals.
X axis label	The label on the X axis of the chart. In normal orientation (not rotated), the X axis is the horizontal axis. For example, **Time** or **Port Number**. This value is also known as the independent variable.

Y axis label	The label on the Y axis of the chart. In normal orientation (not rotated), the X axis is the horizontal axis. For example, **ICMP echo count** or **Lost Packets**. This value is also known as the dependent variable.
Use legend	Check **Use Legend** to display a legend that displays the symbols and color-coding used on the chart. iTest creates a unique color and symbol (dot, diamond, square) combination for each series. For example, variable1 data points are represented by blue diamonds and variable2 data points are represented by red dots. The Legend displays the list of variables being plotted and their associated symbols.
Y axis must include zero	Specifies a value of zero for the origin of the Y axis. This is useful to expand the scale when charting a variable with a small absolute value and a low range.
Highlight Line Chart Data Points	Toggle to display round dots for each data point on line charts. Default: Not selected • Select to display round dots on the line chart for each data point. • Unselect to display the line chart without the round dots for each data point.
Max item count	Sets the maximum number of data points allowed in the chart. Once the count of data points meets the limit, the oldest data points are deleted from the chart as the newest values are added. As a result, the chart displays only the newest data. This is useful for long-running tests.
Series name	The name of the variable being plotted as it should appear in the legend. See **Use legend**.
X value expression	The expression representing the values that are plotted on the X axis. By default, this is time. iTest automatically scales the time scale as needed. If you are charting the value of a single variable as a function of time, then you do not specify a value for the **X value expression** property. If you are charting one extracted value as a function of another, then specify the expression representing the values that are plotted on the X axis (the independent variable). For example, if you are charting packet loss as a function of port number, then specify the expression that represents the port number, for example **$port**.
Y value expression	The expression representing the values that are plotted on the Y axis. If you are charting only one value, then this is the value extracted by the analysis rule.
Export image width	Charts are exported as JPEG files when you export a test report into HTML. Specify the width of the JPG images in pixels. Default: 1000
Export image height	Charts are exported as JPEG files when you export a test report into HTML. Specify the height of the JPG images in pixels. Default: 500

Processors > Chart Using Bar

The properties that you set in this section are represented in summary form in the **Details** cell of the analysis rule. You can edit the settings in either place.

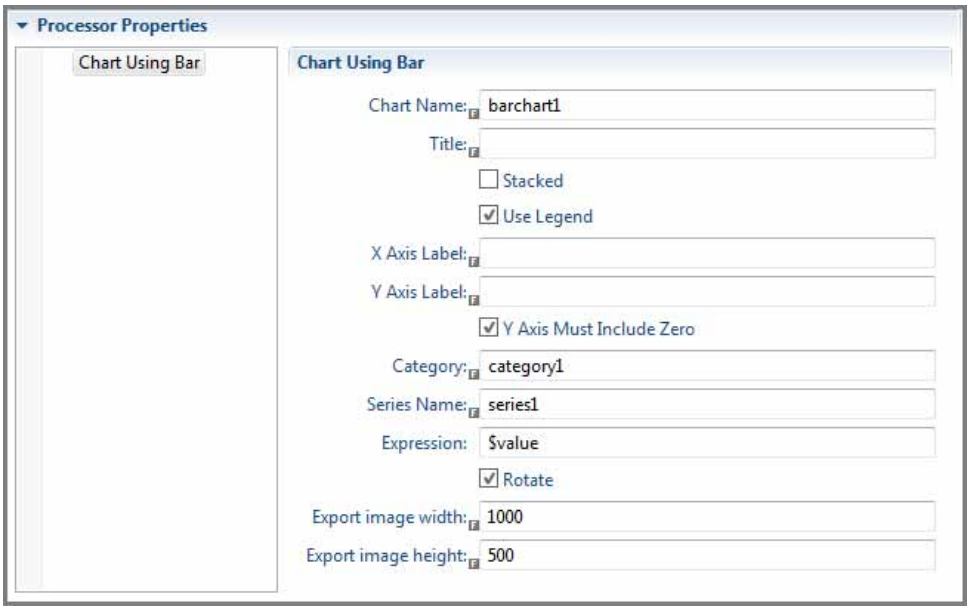


Chart name	The label that analysis rules use to refer to the chart. The label is ***not*** the Title of the chart. Instead, analysis rules from several steps and/or several analysis rules from a single step can send data to a particular chart by specifying the same Chart name. If only a single value is charted on the chart, then Chart name is unimportant.
Title	The text title that appears at the top of the chart when you print or view it.
Stacked	Checked **Stacked** only if you are charting more than one value. When checked, values are added to the chart
Use legend	Check **Use Legend** to display a legend that displays the color-coding used on the chart. iTest creates a unique color for each series. For example, Variable1 data points are represented by blue diamonds and variable2 data points are represented by red dots. The Legend displays the list of variables being plotted and their associated symbols.
X axis label	The label that should appear for the X axis (independent variable) of the chart. For example, Port Number.
Y axis label	The label that should appear for the Y axis (dependant variable) of the chart. For example, Dropped Packets.
Y axis must include zero	Specifies a value of zero for the origin of the Y axis. This is useful for expanding the scale when charting a variable with a small absolute value and a low range.

Category	Specify the expression for the extracted value that distinguishes a particular individual bar. One bar of each **Category** appears along the X axis. The bar is named using the associated **Expression** property setting. If you are charting one extracted value as a function of another, then specify the expression representing the values that are plotted on the X axis (the independent variable). For example, if you are charting packet loss as a function of port number, then specify the expression that represents the port number, for example **$port**.
Series name	A series is the set of values associated with a single variable. The **Series name** is the label for the data as it appears on the chart. All **Series names** appear in the legend with its associated color. See **Use legend**.
Expression	The variable name to plot on the vertical axis of the chart. Usually this will be a value extracted from the response.
Rotate	In the default chart orientation, bars appear to increase vertically. Check Rotate to cause bars to be displayed horizontally and increasing to the right. When a chart is rotated, each additional bar is added further in the downward direction.

Processors > Chart Using Pie

The properties that you set in this section are represented in summary form in the **Details** cell of the analysis rule. You can edit the settings in either place.

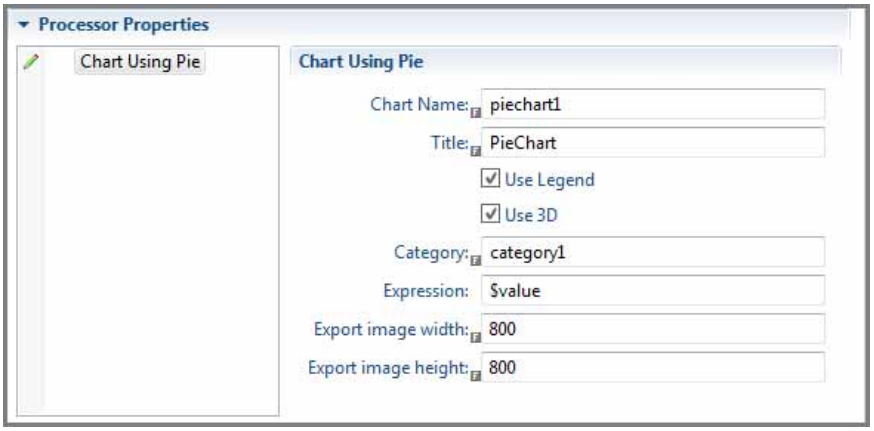


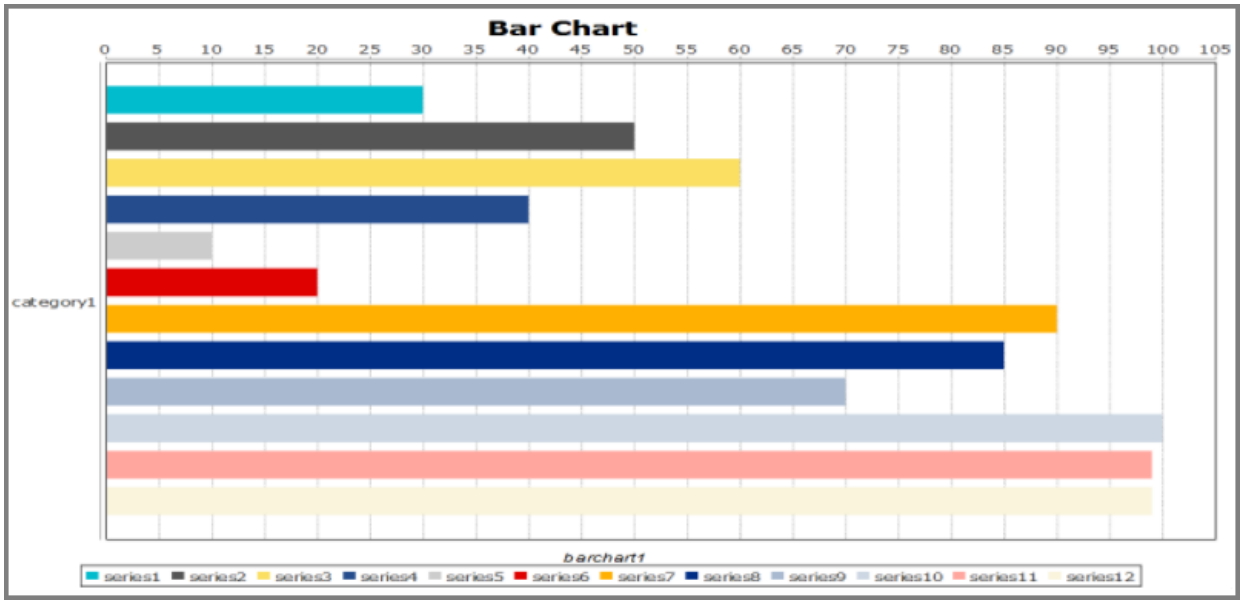
Chart name	The label that analysis rules use to refer to the chart. The label is ***not*** the Title of the chart. Instead, analysis rules from several steps and/or several analysis rules from a single step can send data to a particular chart by specifying the same Chart name. If only a single value is charted on the chart, then Chart name is unimportant.
Title	The text title that appears at the top of the chart when you print or view it.
Use legend	Display a legend that shows the color coding for each Expression that appears on the chart.
Use 3D	Display the pie chart tipped back in a three-dimensional view. This setting does not change the function of the chart, just its appearance.
Category	
Expression	The value extracted from the response to chart.

Creating bar charts

Bar charts display rectangular bars whose lengths are proportional to the magnitudes or frequencies of what they represent. Bar charts are used for comparing two or more values. iTest can display the bars vertically (default) or horizontally.

Specify a setting for each of the following properties for each value to add to the chart (as described in "Specifying the appearance of charts: Chart properties" on page 832):

- Chart name (all values that specify the same setting for the Chart name property will appear on the same chart)
- X axis label
- Y axis label
- Category
- Expression

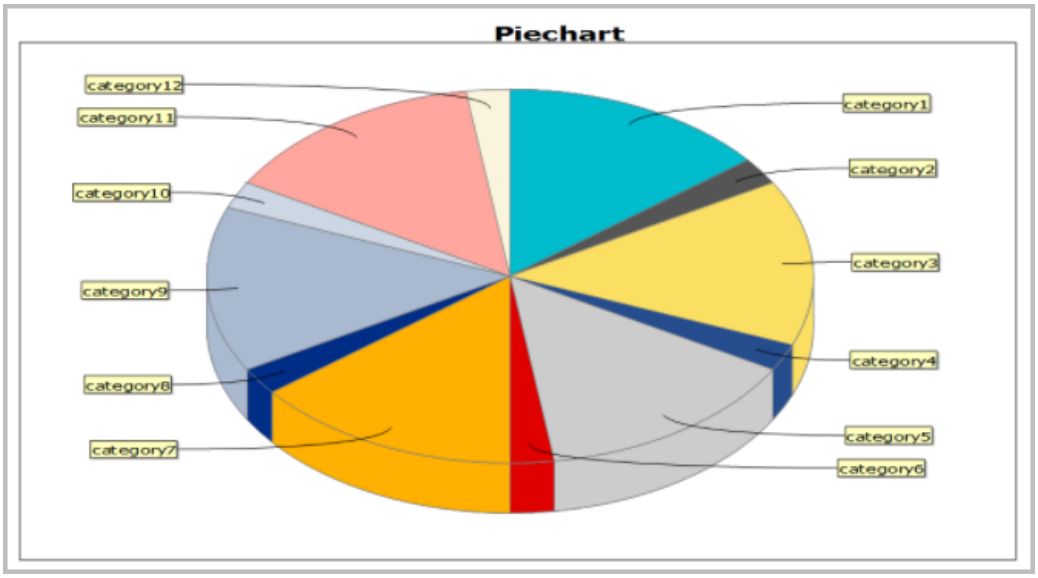


Creating pie charts

In a pie chart, the area of each “slice of the pie” is proportional to the quantity it represents. Pie charts typically display values of multiple variables as a portion or slice, and the sum of all of the values is represented by the full pie or circle.

Specify a setting for each of the following properties for each value to add to the chart (as described in “Specifying the appearance of charts: Chart properties” on page 832):

- Chart name (all values that specify the same setting for the Chart name property will appear on the same chart)
- Category
- Expression



Exporting chart data

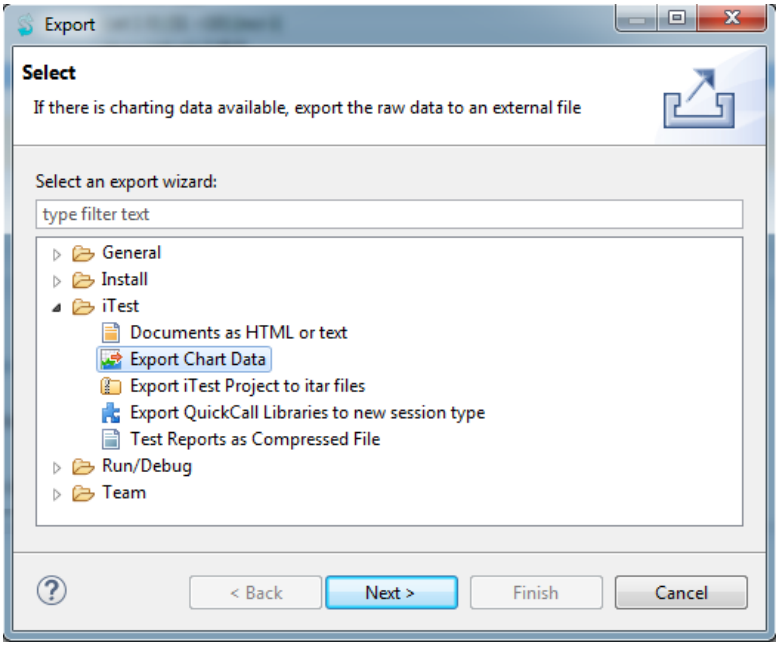
You can export chart data into a standard format file (XML or CSV) so you or others can view the data using other applications (for example, Microsoft Excel or another data visualization application).

Note Neither iTestRT or itestcli can export chart data directly. There are, however methods for saving chart data. See "Exporting chart data from iTestRT or itestcli" on page 841.

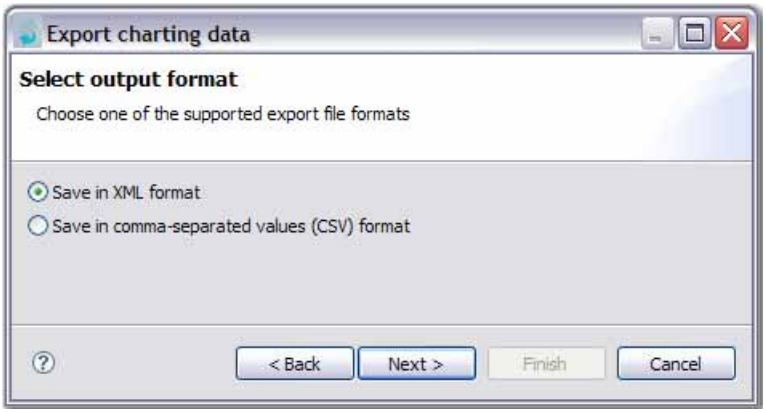
1 Execute the test case that generates chart data.

2 Click **File > Export**.

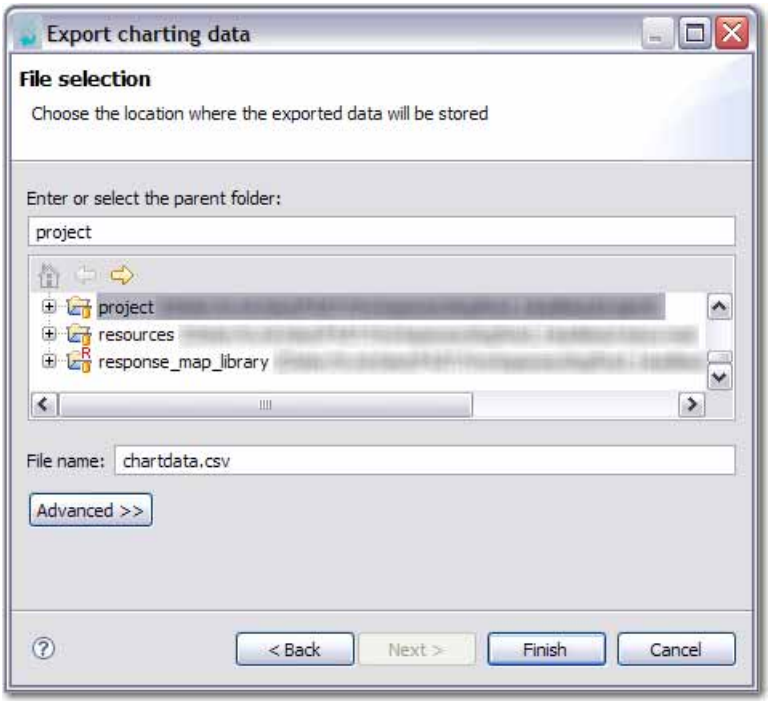
3 On the **Export** page, select **iTest > Export Chart Data**. Click **Next**.



4 You have the option to export the data in XML format or as a CSV (comma-separated value) file. (Many applications can import CSV files.) Click **Next**.



5 Now specify where to save the data and provide a filename. Click **Finish**.



Exporting chart images in test reports

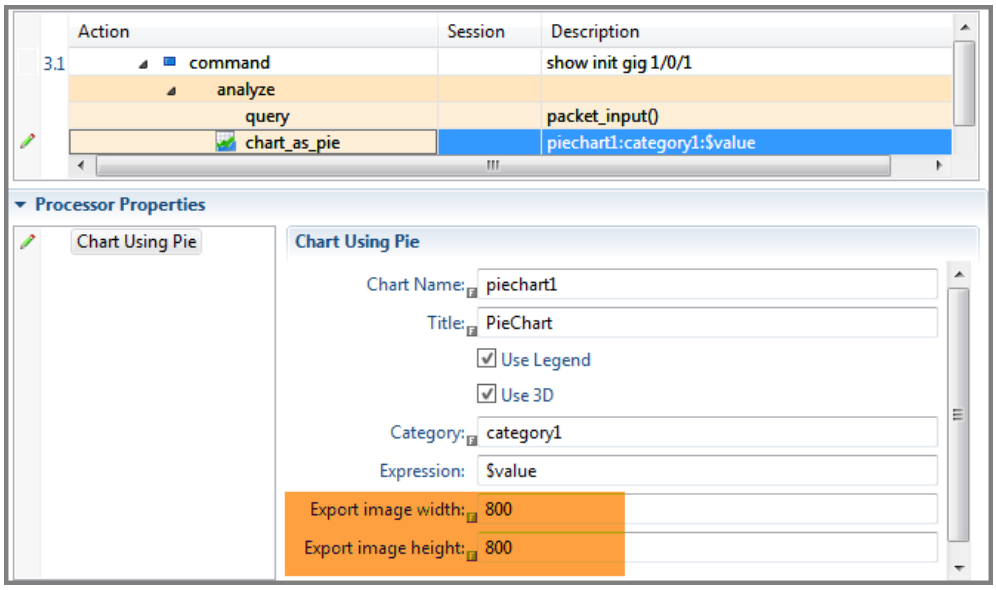
When you export test reports as HTML, iTest includes all charts as JPEG images in the first step of the test case.

The **Export image width** and **Export image height** properties specify the pixel dimensions of the exported JPEG image. The properties appear on the following property pages for analysis rule processors:

Chart Using XY

Chart Using Bar

Chart Using Pie (example:)



Exporting chart data from iTestRT or itestcli

Neither iTestRT or itestcli can export chart data directly. You can use one of the following methods to save charting data:

Create a CSV file using the writeFile command (preferred)

1. Use the **writeFile** command in the test case to write extracted data to a file in CSV format.
2. Use Microsoft Excel to chart the data.

Create custom XSL to extract charting data from a test report

Because charting data is present in test report (XML format files) you can customise XSL to extract the charting data.

Charting one value as a function of another value (XY charts)

In "Charting test case data" on page 829, you specified a single variable whose extracted value changed over time. Time was the independent variable and appeared on the X axis. To chart one extracted value as a function of another extracted value, you specify the independent variable as the **X value expression** property.

For example, if you are charting packet loss as a function of port number, then specify the expression that represents the port number, for example **$port** as the **X value expression** setting.

Setting preferences for Charting View

To view or edit settings that control charting view, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Views > Charting View.**

General information on setting and sharing preference settings appears in Chapter , “Configuring iTest Preferences”.

Spirent > Views > Charting View

When executing, open the Charting view if charts are available	Select to ensure that the Chart, if any, displays when your tests execute.
Chart color	Select the color palette that will be used to display the charts. The Chart color drop-down list provides 10 color palettes and each palette has 10 colors. Options: Beach, Bermuda, Fall, Gentleman, iTest, Marine, Party, Playground, Vacation, Velocity Default: iTest Each selected color palette consists of 10 colors. If your chart needs to plot more than the colors on the selected color palette, iTest uses colors from the next color palette. For example, if you select, iTest as your chat color palette, and your chart needs more colors to chart, then, iTest will pick colors from the next color palette (iTest → Marine→ Party→ Playground→ Vacation→ Velocity → ....→iTest ....).

CHAPTER 41

Configuring iTest Preferences

Setting iTest preferences

You can set a significant number of preferences that configure iTest editors and views and that change how iTest behaves.

Note The topics that describe most preference settings appear in the appropriate chapter. For example, the chapter on Process sessions includes a topic on setting Process session preferences. This chapter describes the more general settings.

You can share preference settings with other iTest users. See "Exporting and importing iTest preference settings" on page 861.

To specify a preference setting:

1. Click **Window > Preferences**.
2. On the **Preferences** dialog box, in the **Preferences** tree, click **Spirent** and then navigate to the appropriate page.

Note On the **Preferences** dialog box, in the **Preferences** tree, click **General** to list the Eclipse default settings, navigate to the appropriate page and set up preferences as required.

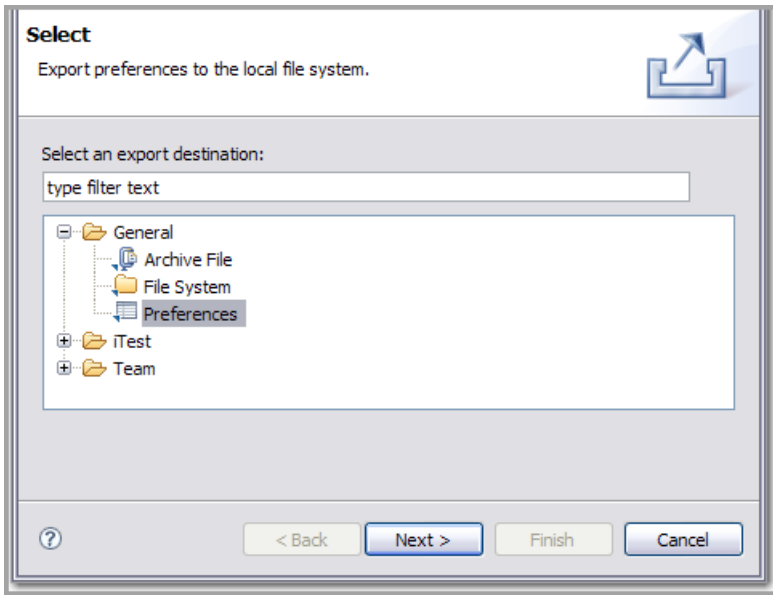
For example: Click Editors and change the value of **Number of opened editors before closing** as required. The default value is **25** and maximum is **99**.

Exporting and importing iTest preference settings

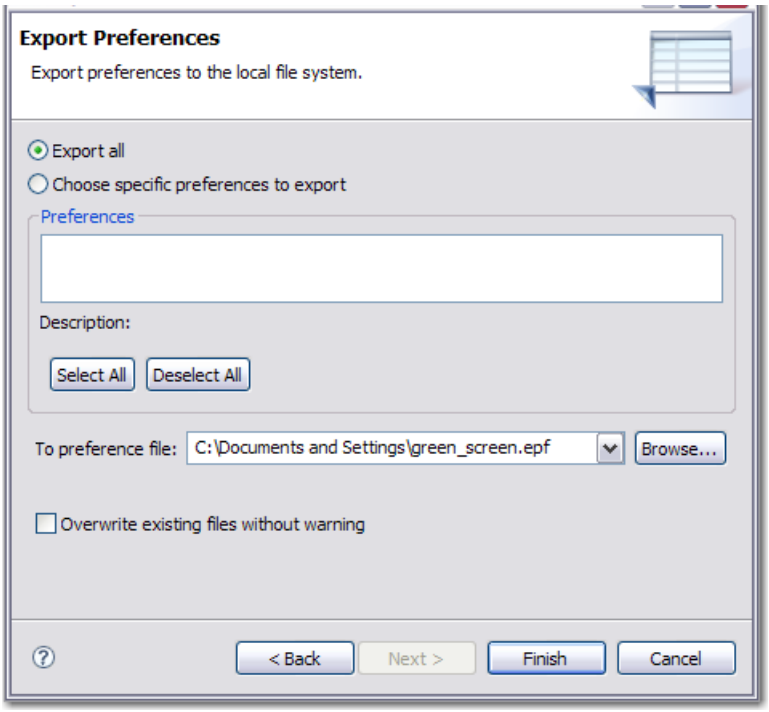
You can export the entire set of iTest preference settings or a specified group of settings so that a coworker can import the settings.

1. Click **File > Export**.

2 On the **Export** page, select **General > Preferences**. Click **Next**.



3 You have the option to export all preference settings or any particular settings that you specify. Browse to a location in the file system and provide a name for the output file that will hold the settings (it will have an **.epf** filename extension). You have the option to overwrite existing preferences files. Click **Finish**.



Preferences: Spirent > Editors

Show warnings encountered when loading a file.	While loading iTest documents, display warnings. Default: checked

Preferences: Spirent > General > General preference settings

Test Case preferred language	Select the default language that will be used to create Test Cses, Session Profiles, and Topologies. Options: Tcl, Python Default: Tcl

Preferences: Spirent > General > Appearance

Show buttons with text (setting takes effect after you restart iTest)	Several buttons include descriptive text in addition to the typical icon. Uncheck the box to display only the icon in the button. Default: checked

Preferences: Spirent > General > Code Review Integration

Spirent iTest allows you to configure settings that controls text representation for code review.

Enable text representation for code review integration	Select this option to save iTest cases as .txt for for code review and integration. Default: not selected
Apply code review integration for the following file types	This option is availalble only when **Enable text representation for code review integration** is selected. iTest allows you to select the following file types as .txt files: Testcase (.fftc), Testbed (.fftb), Session Profile (.ffsp), Topology (.tbml), Parameter (.fftp), Response Map (.ffrm). Defaut: All file types selected. Clear selection if you do not want to save file type as .txt file.
Hide text representation files in workspace	When selected (default), the text files are hidden in a workspace. Clear selection to display text files in a workspace.

Preferences: Spirent > Log Settings

Spirent iTest captures the real-time data and writes a log file according to the logging level selected to a specified location. You can view this log file during a test run and then analyze when the test concludes.

Logging Level	
DEFAULT	Indicates the log message to be recorded in log file and inserts a comment into the Log file.

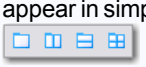
ERROR (Errors only)	Indicates only error message are to be recorded in the log file. The Error Log displays exceptions. When working with Customer support, email the contents of the log to help resolve the issue.
INFO (Warnings, errors and information messages)	Indicates the type of message recorded in the log file. The Log displays warnings, exceptions, and information messages. When working with Customer support, email the contents of the log to help resolve the issue.
DEBUG (Message useful for debug iTest)	Indicates only debug messages are to be recorded in the log file. The Log displays debug messages. When working with Customer support, email the contents of the log to debug any issues with iTest.
OFF (Turn off logging)	Indicates that no log messages are to be recorded in the log file.
Log file location **Note** You may view log by clicking menu: **Windows > Show View > Error Log**	
Auto-select	Indicates the location where the log file is stored. Default: Selected
Use local file	Select to store the log file in the desired location. Click **Browse** to navigate to a desired location, enter the name of the log file to be saved. Default: Not selected Value: C:\Users\user-01\workspace\iTest.log

Preferences: Spirent > News

You can view news updates from Spirent as they arrive or can view them on-demand by navigating to this page and clicking **Show iTest News.**

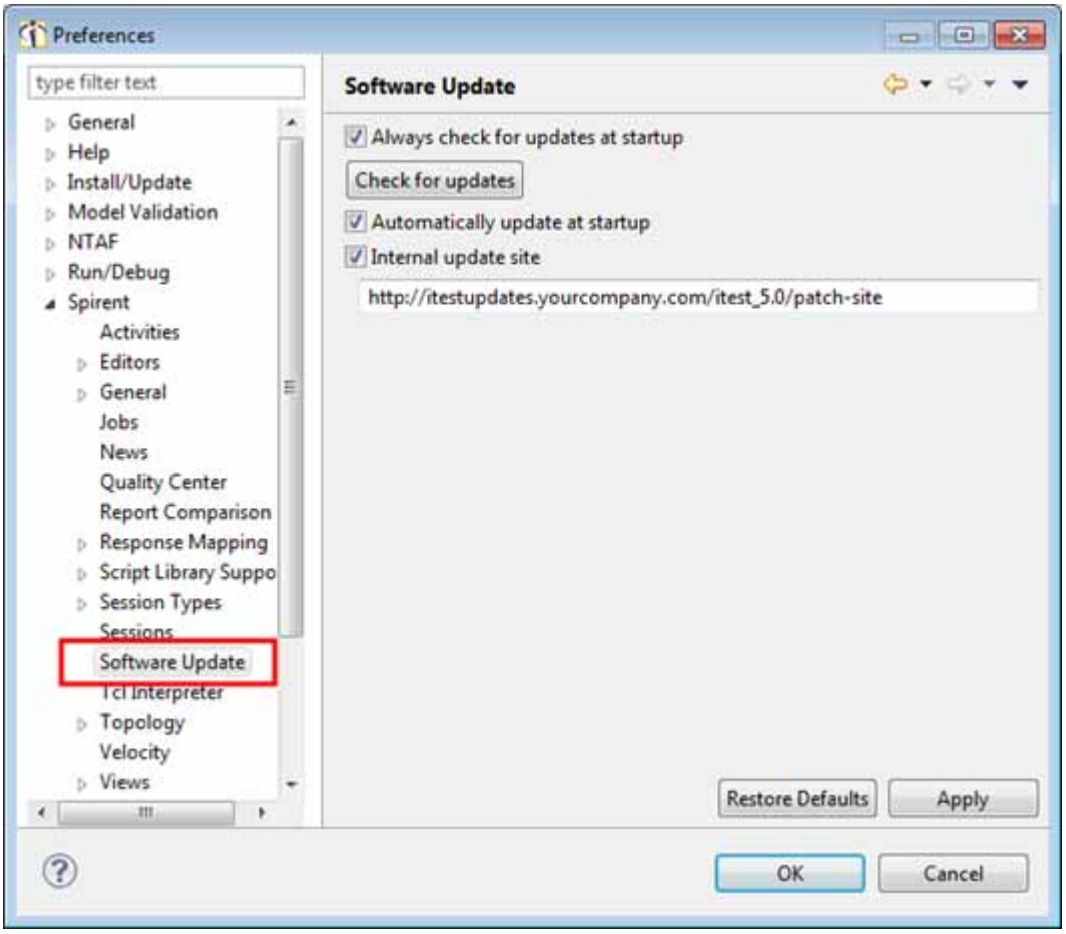
Do not show iTest news	Uncheck the box to display a dialog box that notifies whenever Spirent sends a news update. Default: checked
Check for news updates	Check the box to have iTest regularly determine your installation configuration and send appropriate iTest news updates. Default: checked

Preferences: Spirent > Sessions

Session Start Mode	Default: **Start in a compatible session window** **Start in current window**: Start each new session in the iTest window, and not in a new session window. You can use the window management buttons  to layer or tile the sessions. **Start in a compatible session window**: Start each new session in a session window of the same general type. For example, any CLI session type can open in the session window of any other CLI session type (they are all terminal-based sessions that typically appear in simple text windows). You can use the window management buttons  to layer or tile the sessions. **Create a new window for each session**: Start each new session in a new session window. As a result, one session window will open for each new session
Display a warning ... and **Always open the 'Execute a QuickCall' wizard ...**	These property settings are associated with iTest QuickCalls. See "QuickCall preference settings" on page 218 for details.

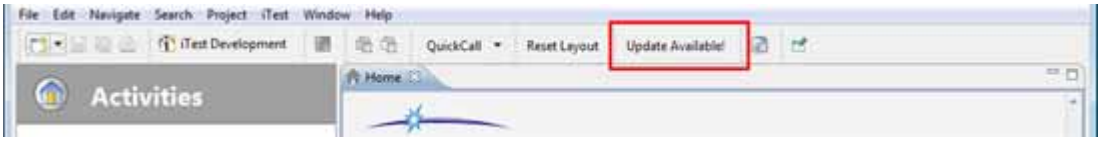
Preferences: Spirent > Software Update

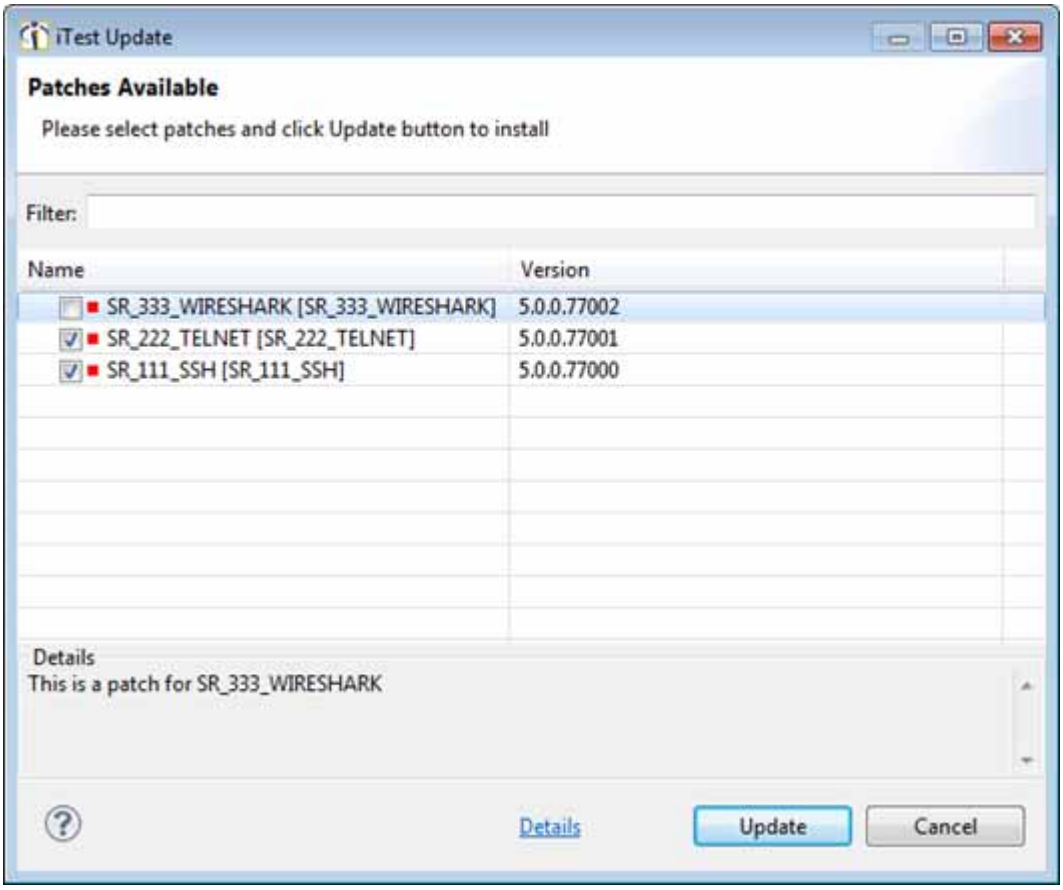
iTest allows you to configure automatic Software updates via **Preferences > Spirent > Software Update** page.



Always check for updates at startup	Select to check for updates on Spirent update site **When Not selected:** When **Always check for update at startup** is not selected, the **Automatically update at startup** is disable and does not affect any updates. **Default:** Selected. **Note** Checks the internal customer's site if the you have selected the **Internal update site** option.
Check for updates	Select to check for updates automatically at startup. **Default:** Not selected

Automatically update at start up	**When selected:** iTest installs available updates at startup automatically (that is, without your conformation). **When Not selected:** When updates are available and the **Automatically update at startup option** is not enabled, iTest displays an **Updates available!** button on the workbench to indicate that updates are available. (see illustration below). 1. Click **Updates available!** and iTest opens a **Patches available** dialog that lists the available updates. 2. Select and install the required updates manually. **Default:** Not selected
Internal update site	The **Internal update site** option is provided for customers with security restrictions or controlled environment. Select the option and input the appropriate location. **Default:** Not selected





Preferences: Spirent > Tcl Interpreter

iTest uses a Tcl interpreter during execution. There is no need to install a Tcl interpreter if you do not already have one because, by default, iTest follows this process when determining which Tcl interpreter to use:

- Use the first interpreter found in the system PATH variable. If not available, then:
- Use the interpreter specified for the **Use the specified Tcl interpreter** property. If not available, then:
- Use a built-in interpreter (base on JACL).

If, however, you want iTest to use an external Tcl interpreter (one installed on the same computer as iTest), we recommend that you configure the system PATH to include the preferred Tcl interpreter, rather than specifying the path in this preference.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: 1. If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. 2. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. 3. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Built-in**: Use iTest's internal JACL Java-based Tcl interpreter. Because JACL does not support any C/C++ extensions, most traffic generator devices will not work in this interpreter. (If you need to specify a particular Tcl interpreter for your device, find the device's software in the **Session Types** properties group in this topic, for example, **Session Types > Ixia Traffic**). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: **Auto-select**
Log Tcl commands to a console	Log all commands to the Tcl interpreter to a console. Default: unchecked
Log Tcl responses to a console	Log all responses from the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

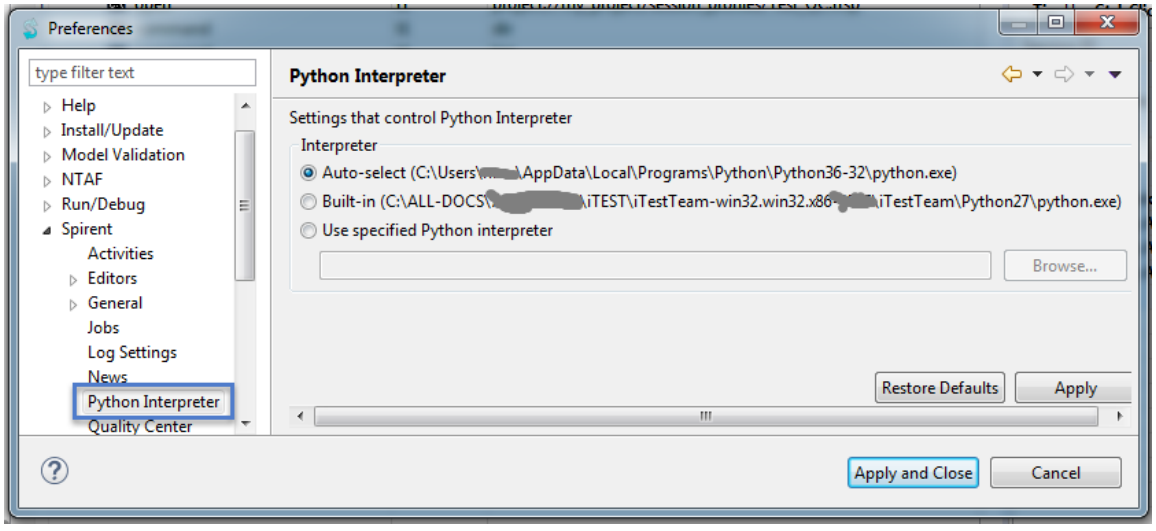
Preferences: Spirent > Python Interpreter

iTest uses a Python interpreter during execution, if set as Test Case preferred language. See "Preferences: Spirent > General > General preference settings" on page 863. There is no need to install a Python interpreter if you do not already have one because, by default, iTest follows this process when determining which Python interpreter to use:

- From "PATH" environment variable
- From Windows Registry on Windows
- From default paths of installation in Python installer

If, however, you want iTest to use an external Python interpreter (one installed on the same computer as iTest), we recommend that you configure the system PATH to include the preferred Python interpreter, rather than specifying the path in this preference.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest looks for python installed in the in default paths. **Built-in**: ITest looks for Python isntalled in the iTest installation folder. Built-in Python interpreter is available only for Windows platform, since Linux, by default installs Python 2.7. **Use the specified Python interpreter**: Select this option. Specify a particular version of Python interpreter in the text box or browser to the location of the particular Python installation and select Use this option only if your application must use a particular interpreter. Default: **Auto-select** Click **Apply/Apply and Close**



CHAPTER 42

Chat Sessions (XMPP chat)

Sending and receiving XMPP chat messages during test execution

Capabilities

You can add steps that receive and send XMPP chat messages during execution. A test case can send and receive as many messages as are needed. Message text can contain both fixed text and response data.

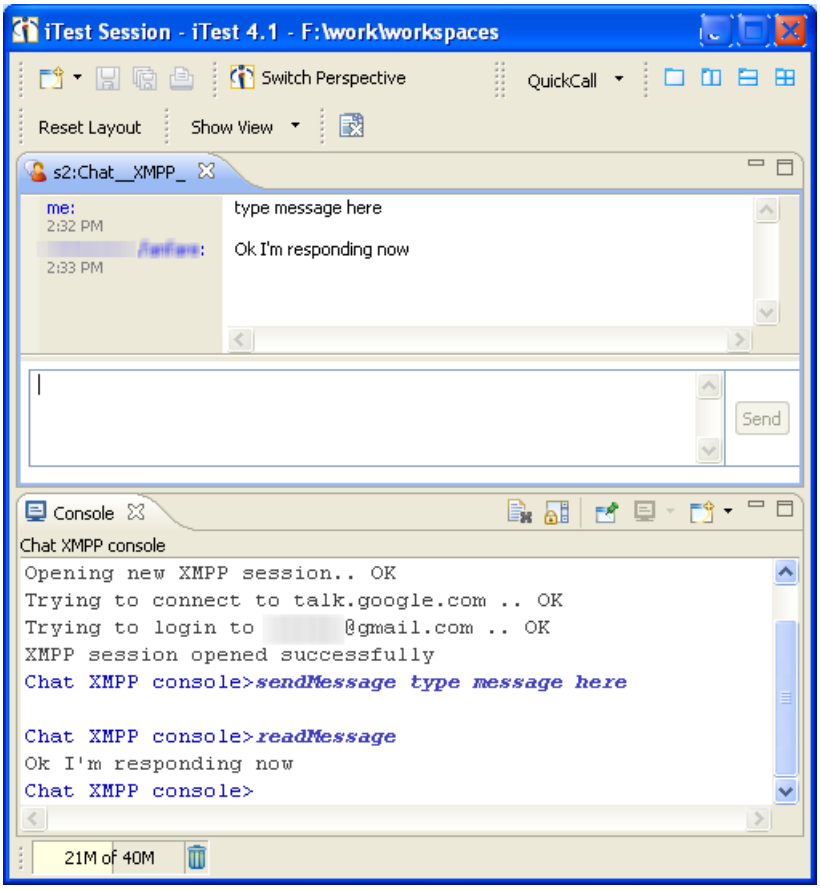
- You can use field replacements to insert variable values and parameter values into the message.
- Chat steps that generate errors set the test case result to **Fail** and continue execution.

Example uses

- A network protocol test that runs configuration steps, drives traffic across the network, and then pauses execution and sends you a chat message. You then physically connect/disconnect cables and then send a chat message to the test. The test then continues to execute and verifies whether routing is configured correctly.
- After many steps in a complex automated network test case, the test pauses execution and sends you a chat message. While execution is paused, you investigate the state of the network by starting interactive Telnet and/or SSH sessions with particular devices. You then decide which path the rest of the test should use. Based on your findings, you send a chat message that supplies (pre-coded) information to the test case. Based on the information in the message, the test case continue execution down the path you specified.
- A complex test case typically runs smoothly, but in some rare conditions it runs into an error and fails unexpectedly. You add steps that pause execution when the failure occurs and that send you a chat message to notify you of the failure. You can then check the state of the network and the state of test execution. Once you determine what went wrong, you can send a chat message with pre-configured content to either cause execution to continue or terminate.

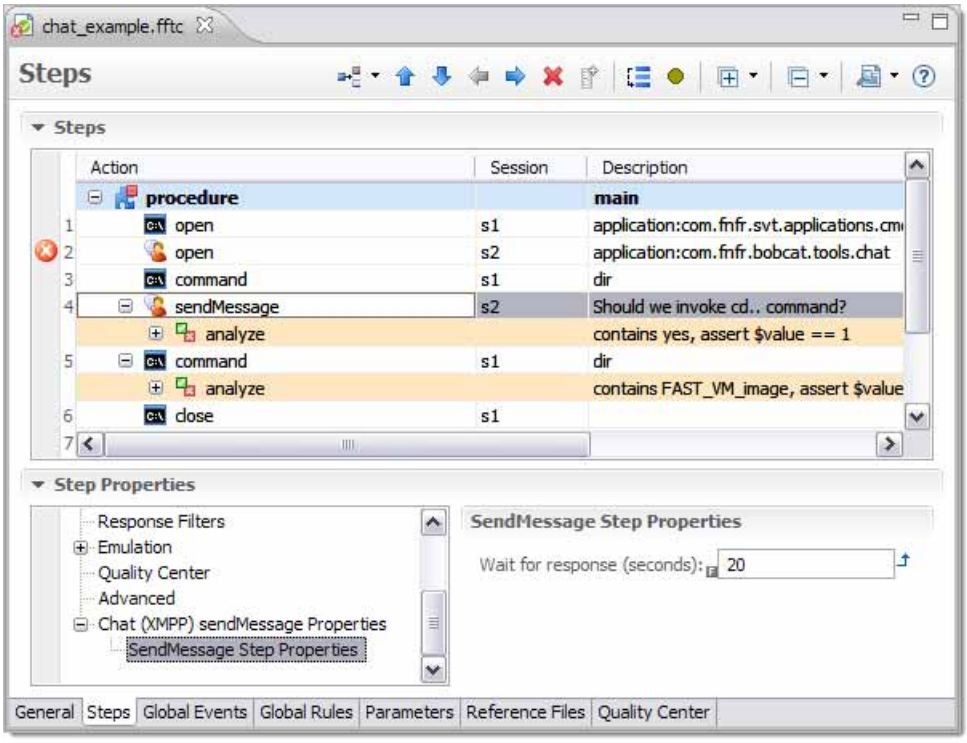
Creating Chat test cases

The easiest way to create chat steps in a test case is to save a captured Chat (XMPP) session to the test case. Here is a a iTest Chat (XMPP) session:



Here is an example Chat test case:

As with any test case step, you can modify the step properties as needed. For example, you could add field replacements to include response data in **sendMessage** text.



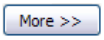
Chat action reference

flushMessages	Deletes the messages that are currently in the iTest message queue. Use this action to prepare for a **readMessage** step to ensure that the **readMessage** receives the next message from the sender and not a message that is currently in the queue. No configurable step properties.
sendMessage	Sends the current message text. Type the message into the **Description** cell. Field replacements are supported. To supply multiple lines of text, in the **General** property group, for the **Command** property, click **Details** . Type the text into the **Command** text box. **Step properties: Chat (XMPP) sendMessage Properties** **Wait for response:** Specify the maximum time to wait in seconds for a response. To specify no time limit, specify a negative integer. Default: 20
readMessage	Read the most recent message in the iTest incoming message queue. See the **Use pending response** step property. **Step properties: Chat (XMPP) readMessage Properties** **Use pending response**: Check the box to read the most recent message in the iTest message queue. Uncheck to ensure that the **readMessage** receives the next message from the sender and not a message that is currently in the queue. See the **flushMessages** property. Default: checked **Wait for response**: Specify the maximum time to wait in seconds for a response. To specify no time limit, specify a negative integer. Default: 10

Session profile property settings for Chat (XMPP) sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Chat session profile properties

To	Specify the address to connect to.
Use default XMPP connection	Check the box to specify that **Velocity** should act as the XMPP server. Default: checked
XMPP Server	Specify the XMPP server address: either the DNS name or IP address.
Port	Specify the port that the XMPP server should listen on. Default: **5222**

Login name / Password	Specify the username and password of the Chat account that will send your messages.
Resource	Specify the identifier for the Chat sender for the message. This enables you, for example, to specify a value of **Home** for messages that you send from your home computer and specify a **Resource** of **Work** so that you can send and receive messages using the same chat account in both locations.

Capture Replay

Treat first received message after send as response	Uncheck the box to ignore the first incoming message. Default: checked
Capture received messages	Check the box to cause iTest to capture all received messages. In some cases, you may want to speed processing and shorten reports by not capturing responses. Default: checked
Default response timeout	Specify the maximum time to wait for a response. The setting enables test cases to succeed even when the receiver is offline at execution time: The test case can send a message to an endpoint that is offline and wait until the receiver responds (or until the timeout expires) before it continues to execute at the next step. To specify no time limit, specify a negative integer. Default: **10**
Ignore incoming timeout	Specify the amount of time to wait before capturing responses. The setting enables you to ignore a first incoming message, for example, if it is of no use in the test. To specify no time limit, specify a negative integer. Default: **0**

Display

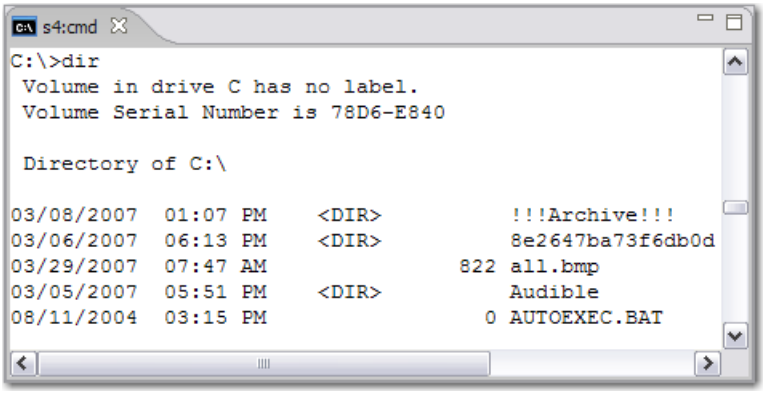
Wrap test at column	Specify the maximum number of characters to display on a line.

CHAPTER 43

Command Prompt sessions

Command Prompt session window (Microsoft Windows Command Prompt)

In a Command Prompt session, you can execute commands in the Windows Command Prompt shell. The Command Prompt session window displays your commands and the local PC's responses. .



Typical uses:

- Perform file-related activities like copying, renaming, deleting files and directories
- Run a command-line utility to perform some action and/or retrieve some information
- Start a process, batch file, or application

Tips

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and Paste them into an active session. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step.

Unsupported features

Note Interactive commands are not supported. For example, Start-Service, Stop-Service, Restart-Service, etc., actions that read from console are not supported.

iTest starts a Command Prompt process and controls it though standard in and standard out. Therefore, the following Command Prompt features are not supported:

- Tab completion
- 'More' page continuation (iTest responds to both multi-screen responses and piping results to more with a single page of contiguous output.)
- Applications launched from within Command Prompt that attempt to take over the Command Prompt window (for example, Telnet)

Note Running GUI applications directly from the Command Prompt session window (for example, by typing notepad) may result in a significant delay in displaying the GUI window. To avoid the delay, use the **start** command to launch other applications from the Command Prompt session (especially GUI applications). For example, to start Notepad, type:
`>start notepad.exe`

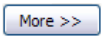
Note for Linux and Unix users:

Because Linux and Unix devices typically include installed Telnet and SSH servers, you can start a Telnet or SSH session to **localhost** to access a shell.

Session profile property settings for Command Prompt sessions (Microsoft Windows command line)

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Note It is recommended to specify the prompts before starting a new session. If you start a session without prompts, it is required to write at least one command to ensure that the rest of the commands that follow will be captured correctly.

Note for Linux and Unix users:

Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet or SSH session to "localhost" to access a shell.

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Prompts

For an overview on how prompts work, see "Overview: Prompts in iTest" on page 463.

For instructions on using the properties in this group to define prompts, see "Editing prompt definitions" on page 467.

For related prompt properties, see "Terminal > Replay > Step Defaults > Completion" on page 884.

Name	**Note** This property setting has no effect for Command Prompt sessions.
Content	Specify the exact text of the prompt. **Note** All prompt definitions are case-insensitive and leading and trailing whitespace is trimmed from any prompt text before iTest attempts to determine whether response text is a prompt. If you use regular expressions in the **Content** value, then set the **Type** property to **Regex**. If the prompt includes a space character or any whitespace in the body of the text, be sure to set the **Type** property to **Wildcard**. Default: [none]
Type	Specify the kind of prompt. **Normal**: Interpret the text in the Content field as the case-insensitive text that you expect for the prompt. **Wildcard**: Disregard any characters that appear in the location of the * character in the text specified for the **Content** property. The most common application for the **Wildcard** setting is to allow for leading or trailing numeric or UserID characters in the prompt (for example Device02>, Device03>, and so on). If you set **Type=Wildcard**, then only the * wildcard character is allowed within the **Content** string (and no other wildcard characters like ?). To use other wildcard characters in the **Content** string, you must use **Type=Regex**. **Regex**: Interpret the text specified for the **Content** property as a regular expression. Default: [none]
Is more prompt **More next command** **More quit command**	The **-- more --** prompt is a common method for allowing command line users to view one screen (page) at a time. Many devices use the space character as the command to move to the next page (and often, the letter **q** to exit the display of the response). To enable your automated test cases to page through data that is displayed one page at a time, iTest can automatically "press the space bar" as often as is required to get to the end of the response. As a result, the device's response to the command becomes a single uninterrupted flow of text that does not include the More text. If the prompt is a page-control prompt (for example - - more - -, then: 1. Select the **Is More prompt** checkbox. 2. In the **More next command** text box, specify the command characters (typically a space character) that cause the next page to appear. By default, a space character appears in the box. 3. In the **More quit command** text box, specify the command that exits the More display and returns to the command line prompt. By default, a **q** character appears in the box. 4. Specify a value for **Terminal** > **Replay** > **Step Defaults** > **More**.

Terminal > Replay > Command

Send interval between each character (milliseconds)	**Note** This property setting has no effect for Command Prompt sessions.

Terminal > Replay > Step Defaults > Terminator

Line terminator	**Note** This property setting has no effect for Command Prompt sessions.

Terminal > Replay > Step Defaults > Response

Treat LF as CRLF	**Note** This property setting has no effect for Command Prompt sessions.
Filename to write response to	Specify the URI of a file to write the responses into. You can use field replacements in the text of the URI to allow the test case to set the filename at runtime. For example, `file:subdirectory_name/[param file_to_create]` Default: <none> **Note** The full text of the response is written to the file, regardless of the **Number of lines to keep** setting. See the following associated properties: **Append response to file** **Response header** **Number of lines to keep** **Write echo to file** **Write prompt to file**
Response header	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: You may want to specify a text string that should appear before each block of response text. For example: +-+-+-+-+- Next Response Starts Here -+-+-+-+-+ Default: <none>
Number of lines to keep	For very long responses, you might not want to keep all of the response text (as displayed in the Response view for the selected step while working in the Test Report editor or in the Test Case editor). Specify the maximum number of lines to keep for any single response. Specify **0** (zero) to keep all lines. **Note** If you specify a URI in the **Filename to write response to** property, then all lines in the response are written to the file, regardless of this setting. Default: 10,000
Append response to file	If you save responses to a file by specifying a value for the Filename to write response to property, then: Check the box to append each new response to the file specified in the **Filename to write response to** property. Uncheck the box to replace the text of the file specified in the **Filename to write response to** property with the most recent response. As a result, the file will hold only the last response in the session. See the **Filename to write response to** and **Response header** properties. Default: Checked
Write echo to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: Check the box to include any echoed characters in the saved response. Default: Checked

Write prompt to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: • Check the box to include the last line of the response in the saved response (in command-line applications, this is typically the prompt after the response). • Uncheck the box to not save the last line of the response to the file (the prompt at the beginning of the response where the command was typed is still saved). Default: checked

Note

Options **Write echo to file** and **Write prompt to file** do not work in capture mode. These options are available for **Replay** mode only.

The example below shows how the commands/responses are echoed in these scenarios.

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options *are selected*:

```
prompt>command
response text
etc
etc
prompt>
```

Write echo to file

- When **Write echo to file** is *not selected*

```
response text
etc
etc
prompt>
```

- When **Write prompt to file** is *not selected*

```
prompt>command
response text
etc
etc
```

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options are *not selected*

```
response text
etc
etc
```

This is because the **Terminal > Replay** options are used to replay the captured steps. That is, replay the steps captured via test case execution or replayed from the Capture View ("Working in the Capture view" on page 103).

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXXX.txt**

Terminal > Replay > Step Defaults > More

Pages to fetch	For responses that are longer than be displayed on a single screen, devices often provide a page-control prompt that enables you to view one screen of text at a time (for example **- - more - -**). Specify the number of pages to fetch when the **more** prompt appears (zero means get all pages). If the setting is non-zero, then iTest retrieves that number of pages and then terminates the output from the session's response by sending the command specified for the **More: Quit Command** property. Default: 100
Device does not remove more prompt. Remove more prompt from response	**Note** This property setting has no effect for Command Prompt sessions.
Use BELL character to detect end of more pages	**Note** This property setting has no effect for Command Prompt sessions.

Terminal > Replay > Step Defaults > Completion

You use **Completion** settings to define when the execution of a step should be considered complete. The determination of when a step is complete is protocol-specific. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- For some steps, you might have defined analysis logic to examine the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

For CLI protocols, you can specify any of several conditions to define when the step is complete, for example, the existence of certain text in the response or the time elapsed after sending the command. The default setting of the **Completion criteria** property is that the step is complete when:

a The session channel is idle for the time specified by the **Idle channel interval** property

and

b The last line of the response matches one of the prompt definitions specified for the session profile or device.

Idle channel interval	This setting helps in cases where you do not know what response to expect and can use a specified idle time (for example, 100 milliseconds) or when you expect no response whatsoever, for example, when talking to a terminal server. Default: 100
Wait for first character before starting idle	Some devices do not respond to a typed command immediately. This setting enables you to ignore the idle time after the last character of the command is echoed and the first character of the actual response is returned. This way, the delay is not misinterpreted as idle channel time for the purpose of determining completion. Default: True

Completion criteria	**Prompt matches AND device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property and last line of the response matches one of the prompt definitions specified for the session profile. **Prompt matches OR device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property **or** the last line of the response matches one of the prompt definitions specified for the session profile. The following processing order occurs: The step is completed once one of the defined prompts is received. If none of the defined prompts is received or no prompt is defined, then the system waits for the specified **Idle channel interval** time (during which the device sends no response data) and then completes the step. **Device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property. **Completion time has expired:** The step is complete when the time specified by the **Completion time** property has elapsed. If you specify **Completion time has expired**, then the **Idle channel interval** property setting is ignored. **TL1 End of Message**: For session profiles that will support TL1 devices, see "Configuring sessions and test case steps for TL1 devices" on page 1215 Default:**Prompt matches AND device has not sent data during the Idle channel interval**
Completion time	Specify the time interval that must elapse for the step to be complete. To apply this setting during execution, the **Completion citeria** property must be set to **Completion time has expired**.
Where to find prompt	Specify where the prompt in a response normally appears. **Last line** **Last non-empty line** The **Any line** setting is a special case that you can use to detect a change in state for an ongoing response. For example, you can detect a port's connection status based on whether the first character of a ping response is **u** or **s**. Be sure to use wildcard characters as needed in the **Content** property. Default: Last line
Command to send when a step is cancelled	Specify the characters to send when the user cancels step execution. Default: \03 (Ctrl-C)
Capture only the last screen of response text	Use this property when you expect a very large response, but the only data of interest appears at the end of the response text. Check the box to cause iTest to save only the last screen of response text. Default: Unchecked
Unknown Prompts During Automated Execution For an overview on how iTest recognizes prompts, see "Overview: Prompts in iTest" on page 463. For instructions on defining prompts, see "Editing prompt definitions" on page 467.	

Expected maximum Idle channel interval	The time to wait for a prompt during automated execution. When this time is reached, iTest displays a **Learn this prompt** link in the status bar. • If the user clicks the link, then iTest opens the **Learn Prompt** dialog box to enable you to add the prompt definition. • If the user chooses not to click the link and the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues. Default: 5
Extra wait before alerting user	Additional time to wait for a prompt during automated execution after the **Expected maximum Idle channel interval** time has been exceeded. When (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed, then: • A countdown timer in the status bar starts to count down the **Time for user to respond** time period. • iTest displays a **Keep waiting** link in the status bar. • If the user clicks the **Keep waiting** link, then iTest waits for an additional (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed period. • If the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues Default: 15
Time for user to respond	Specify the amount of time in seconds to wait for the user to respond once the status bar displays the **Waiting for prompt** timer. Default: 30

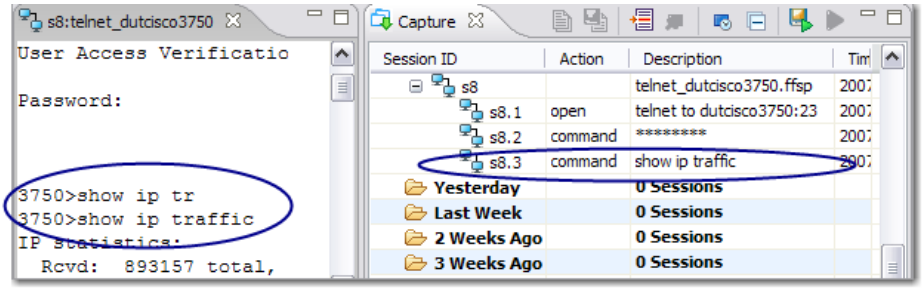
Terminal > Replay > Step Defaults > High Availability

See "Testing HA devices: Detailed instructions" on page 822.

Terminal > Capture

Perform capture cleanup	When you perform manual testing in CLI sessions, you frequently use meta-characters like backspace and up- and down-arrows to correct your typing. In a Capture report, such commands can be difficult to read and understand. If you check **Perform capture cleanup**, then iTest "cleans up" any keyboard shortcuts and removes meta-characters from the captured commands so that the resulting command text appears as if you typed it fully and correctly. See the **Discard command completion steps** property**.** Default: Checked A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, even though we actually typed **show ip tr**<tab>. iTest discarded the intermediate **show ip tr**<tab> form of the command.
Mask unechoed commands	Check **Mask unechoed commands** so that, before creating a Capture report, iTest masks all **Command** property text for which no echo was returned. Check the box to automatically mask passwords. Default: Checked
Remove echo from response	Check **Remove echo from response** to indicate that the device echoes characters typed at the command line. In this case, iTest ignores echoed characters so that the command text is not added to the actual response text. Default: Checked

Remove prompt from response	Check **Remove prompt from response** to save only the response from the session and not the prompt text. We recommend that you do not disable this setting except in rare circumstances. Default: Checked
Use prompts from the session for cleanup	Check the box to use the prompt definitions specified in the session profile document when cleaning up commands. Default: Checked
Discard command completion steps	To ensure that captured commands in the Capture view are easy to understand, iTest, by default, deletes the intermediate command completion text that was submitted while forming a command. A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, and discarded the intermediate **show ip tr**<tab> form of the command. See the **Perform capture cleanup** property. Default: Checked
Learn prompts	If the box is checked, then, when you close a session and iTest has detected a new prompt, the Update Session Profile wizard starts. You can specify particular prompts in the **Terminal > Prompts** properties. Default: Checked
Learn command completion characters	Learn the character code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can). If the box is checked, then, when you close a session and iTest has detected a new command completion character, the Update Session Profile wizard starts. The default completion character is tab. To specify particular characters, configure the **Terminal > Capture > Command Completion** property. Default: Checked
Learn break characters	Learn the character code that the device interprets as a break (so you can manually cancel an executing step). The learned break characters are added to the **Command break characters** property. If the box is checked, then, when you close a session and iTest has detected a new break character, the Update Session Profile wizard starts. Default: Checked. The default break character is Ctrl-C. **Note** To specify particular break characters manually, configure the **Command break characters** property (in **Terminal > Capture > Break**).
Remove line containing more prompt	Check the box so that, when capturing responses, iTest deletes the lines that include the more prompt. Default: Checked



Terminal > Capture > Response

Number of lines to keep	Specify the number of lines in the response to keep in the Capture report. Default: 10,000

Terminal > Capture > Command Completion

Specify the code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can).

Default: tab (\t **)**

To determine the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters.

Limitations of the Record feature:

- For the **Alt** key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”.
- Function keys are not recorded.

Command completion requires ENTER key	Most devices respond immediately with command completion when they encounter one of the characters specified for the **Command completion characters** property. Set this property to TRUE, if the device, to perform command completion, requires that you press ENTER after typing one of the characters specified for the **Command completion characters** property. Default: Unchecked

Terminal > Capture > Break

Number of lines to keep	Specify the character code that the device interprets as a break (so you can manually cancel an executing step). Default: Ctrl-C To add the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters. **Limitations of the Record feature:** • For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”. • Function keys are not recorded.

Terminal > Font

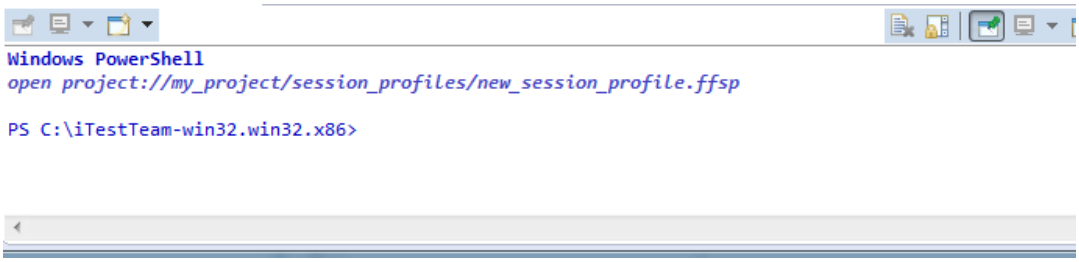
Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

CHAPTER 44

PowerShell sessions

PowerShell session window

In a PowerShell session, you can execute commands in the Windows PowerShell prompt. The PowerShell session window displays your commands and the local PC's responses. .



Typical uses:

- Perform file-related activities like copying, renaming, deleting files and directories
- Run a command-line utility to perform some action and/or retrieve some information
- Start a process, batch file, or application
- Capture test scenarios and support auto-completion and profiles.

Note To ensure that PowerShell opens in iTest, enter the installation path (where you installed PowerShell), in the PATH environment variable. For example, add the following to the PATH environment variable:

C:\Windows\System32\WindowsPowerShell\v1.0;

Tips

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and Paste them into an active session. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step.

Unsupported features

iTest starts a PowerShell Prompt process and controls it though standard in and standard out. Therefore, the following PowerShell Prompt features are not supported:

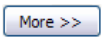
- Tab completion
- 'More' page continuation (iTest responds to both multi-screen responses and piping results to more with a single page of contiguous output.)
- Applications launched from within PowerShell Prompt that attempt to take over the PowerShell Prompt window (for example, Telnet)

Note Running GUI applications directly from the PowerShell Prompt session window (for example, by typing notepad) may result in a significant delay in displaying the GUI window. To avoid the delay, use the **start** command to launch other applications from the PowerShell Prompt session (especially GUI applications). For example, to start Notepad, type:
`>start notepad.exe`

Session profile property settings for PowerShell sessions (Microsoft Windows PowerShell)

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 120. For a detailed description of how iTest uses the settings, see "About property settings" on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Note for Linux and Unix users:

Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet or SSH session to "localhost" to access PowerShell.

PowerShell

.

Profile	Select the profile to be used for the session replay, as required: None, System, Custom None: System: Selecting this uses the default system profile. Custom: Select custom and then select the custom profile you previously created (based on the default session profiles).
Custom Profile	Available when you select Custom Profile. Navigate to the location of your custom profile and select profile to be used for the session replay.

Terminal>Buffer

Buffer: Indicates how much memory to allocate to the Console view. **Note** When the data reaches the limit, the oldest data is deleted.	
Height	The number of lines that the terminal server presents from memory. Default: 140
Width	The number of characters that the terminal server presents on a single line. Default: 3000

Terminal > Replay > Completion

You use **Completion** settings to define when the execution of a step should be considered complete. The determination of when a step is complete is protocol-specific. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- For some steps, you might have defined analysis logic to examine the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

Time to wait for command response (sec)	Specify the maximum number of seconds to wait for a return of response to command. If the response is not received within the specified time, the command times out. Default: 10 **Note** To get a long return of response to command, change the **Timeout to waiting command response**" value to to 60 seconds or more (as required).
Time to wait for Completion response (sec)	Specify the time interval that must elapse for the replay to be complete. Default: 2

Command reference

The following lists a few common commands used. For a complete list of PowerShell commands, go to url: https://technet.microsoft.com/en-us/library/hh848794.aspx.

Note Interactive commands are not supported. For example, Start-Service, Stop-Service, Restart-Service, etc., actions that read from console are not supported.

Get-Command	Gets basic information about cmdlets and other elements of Windows PowerShell
Get-Help	Displays information about Windows PowerShell commands and concepts.
Get-History	Gets a list of the commands entered during the current session.

New-Object	Use to create new object, add members and define object property (for example, add a NoteProperty by giving the new property the name and value that represents the Name property) Example: $my_object = New-Object - TypeName PSObject $my_object | Add-Member -MemberType
	-Name f1 -Value v1 $my_object | Add-Member -MemberType
	NoteProperty -Name f2 -Value 222 Write-Output $my_object
add-content	Use Add-Content command to append data to a text file. To use the multi-line command, add new lines by pressing CTRL+ENTER or SHIFT+ENTER.and then pressing ENTER to exit multi-line mode.

CHAPTER 45

Database Client sessions

Overview: Database Client sessions

The Database Client session window is an interactive browser where you perform database operations and monitor responses. iTest captures all commands and responses and you can save captured items as test case steps that start the session and set and request database records. Automated test cases open the same session window to perform database operations. iTest supports sessions with MySQL, SqlServer, Oracle, Szlite, Derby, or a custom database type that you specify.

iTest saves all responses from the database server as structured data. iTest auto-maps responses and auto-generates queries for response data (that is, blue boxes appear immediately in the Response view when you select a step in the test case). As a result, you do not have to create database server response maps and can immediately write analysis rules that use the auto-generated queries. See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.

Examples of tests that use Database Client sessions

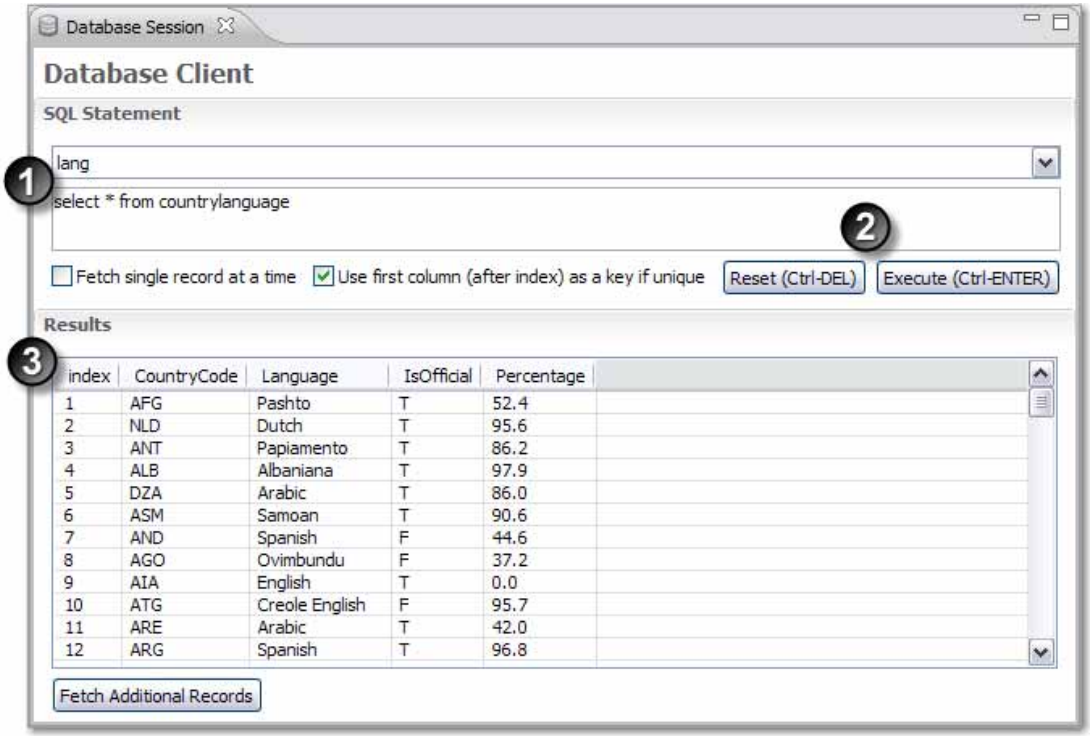
- The system-under-test involves a database and part of the goal of the test is to ensure that the database is populated correctly or that the system-under-test responds properly as data is manipulated in the database. For example, you are working on an NMS test case, and want to peek and poke in an NMS database.
- Data-driven test: You want to pull records from a database and use the data to drive a test case. In this way, one generic test case can cover a lot of different scenarios that can be defined by filling certain information into a database or extracting different information from a database (using a test case parameter in a statement, for example).
- Performance test: You want to harvest data from a test while it is executing and push it into a database for later post-processing

Overview: Data-driven testing

In data-driven testing, you reduce the number of automated test scripts that you design, but at the same time increase the number of test cases that will execute because the steps are defined as records in an external file or a database. iTest supports data-driven testing with Database Client sessions and in several other ways. To learn more, and to see examples, search for “data-driven” on the Community Forums.

Database Client session window

iTest uses the Database Client session window to perform database operations for both interactive sessions and automated test cases.



1 SQL Statement section

In the **SQL Statement** section, you can type any SQL statement that is valid for the connected database. For example, **SELECT**, **INSERT**, **DELETE**, **CALL_PROCEDURE**, **DROP**, and so on.

To help users of database sessions, you can define statement templates in the session profile. The statement templates appear in the drop-down text box at the top of the **SQL Statement** section. When the user selects a statement template, the associated statement text appears in the main text box. The user can then edit and execute the statement. In the example, we selected the **lang** statement template. The associated statement text (**select * from countrylanguage**) appears in the text box, ready for us to execute. (Statement templates are described in “Statement Templates” on page 933.)

Note For a simple statement (for example, **SELECT * FROM countrylanguage**), the column name that appears in the **Results** table is the name that is defined in the database (for example, **isOfficial** as shown in the screenshot). For a statement with an “as” clause (for example, **SELECT isOfficial FROM countrylanguage AS OfficialLang**), the column name that appears in the **Results** table is the alias that is specified in the statement (**OfficialLang** in this case).

- **Notice of update to column names in the database**

  The table column names have been updated to improve their usefulness. In iTest 4.2.0 and earlier, the table column names are:

  index
  COLUMN_NAME
  COLUMN_TYPE
  IS_NULLABLE
  COLUMN_KEY
  COLUMN_DEFAULT
  EXTRA

  In iTest 4.2.1 and later, the names are:

  index
  Field
  Type
  Null
  Key
  Default
  Extra

2 Executing statements

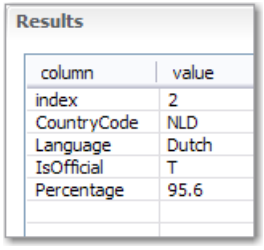
Click **Execute** or press Ctrl+Enter to execute the statement. Click **Reset** to clear the statement from the text box.

3 Results section

The **Results** section displays the results of executing the statement, for example:

- Display a scalar value
- Display the result of executing a **DELETE** statement
- Display a result set. By default, a result set appears as a table containing rows and columns. For table results, the client fetches one page at a time (by default, 100 records). Click **Fetch Additional Records** to view the next page (or as many records as are left in the database). You can specify the number of records per page using the **Records per page** property in the session profile. To go back, execute the statement again.

  If you check the **Fetch single record at a time** property, then each record appears in column / value format. To view the next record, click **Fetch Additional Records**.



Tip If you are creating a step that you will eventually put into a loop, then fetch records one-at-a-time. A record will be returned each time the loop encounters the step

If you check the **Use the first column as a key** property, then iTest auto-generates queries using the key. See “Generating good structured data and queries” on page 928.

- If the statement fails, the **Results** section displays a message describing the error.

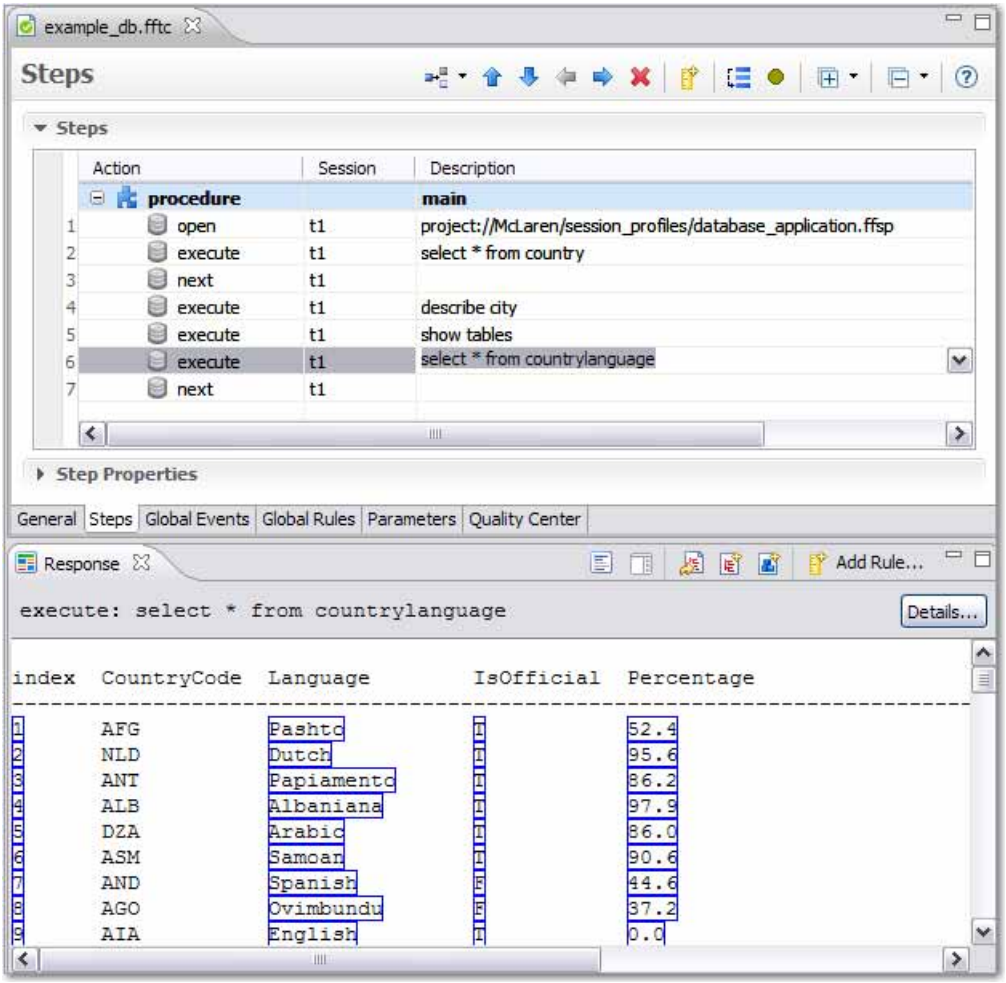
About Database Client test case steps

When you save a captured session to a test case, most steps are captured as **execute** actions with the SQL statement as the **Command** value (displayed in the **Description** cell). When you click **Fetch Additional Records**, iTest captures a **next** action that fetches the next page of records or the next record, depending on the setting for the **Fetch single record at a time** property.

The **execute** and **next** actions are described in “Database Client command set” on page 935.

Notice that iTest auto-maps responses and auto-generates queries for response data (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps and can immediately write analysis rules that use the auto-generated queries.

See “Generating good structured data and queries” on page 928. For background information on analyzing responses, see “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.



Generating good structured data and queries

To make it easy for you to create analysis rules, iTest auto-generates queries for each response. By default, iTest checks to see whether the values in the first column of the response are unique, and if so, iTest uses the first column as the key when generating queries.

Note The **Use first column as key if unique** session property (in the **Execute/Next Step** property group) controls this behavior. See “Session profile property settings for Database Client sessions” on page 929.

Generating queries based on a key

To take advantage of this feature, design statements that return — in the first column — the data that you want to use as the key. For example, in your table (or **JOIN** query) the key column is **customerId** and in the following statement, **customerId** is the first in the projection list

```
SELECT customerId, * FROM customers WHERE sales > 1000000
```

The statement returns all of the rows in the table with the **customerId** column first. This ensures that iTest will use it as the key column when auto-generating queries.

Generating SQL statements dynamically: Using parameters in prepared statements

In some cases, text that you want to use in a statement could create problems. For example, the string is taken from the response to another step and it might contain special SQL characters (like double-quotes) that would make the statement malformed. To solve the problem, you can use a parameter in the place of the problematic string.

Note Do not confuse prepared statements with statement templates.

- **Prepared statements** are statements that include parameter references that will be replaced at runtime with the parameter values.
- **Statement templates** are general-purpose statements that you can select from a drop-down text box while performing interactive testing and then edit as needed (described in “Statement Templates” on page 933).

How a prepared statement works

For any test case step, you can specify that the SQL statement is a prepared statement — a statement that includes references to predefined parameters. For example, instead of using a query like:

```
SELECT * FROM customers WHERE sales > 1000000 AND name LIKE "*Widget*"
```

you can use a prepared statement that looks like this:

```
SELECT * FROM customers WHERE sales > ? AND name LIKE ?
```

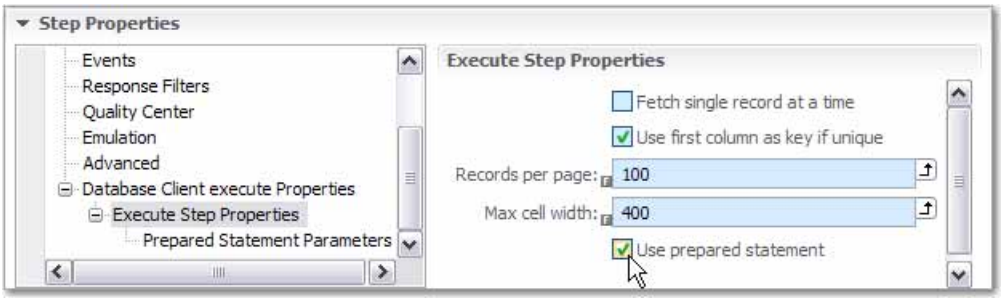
The **?** characters in the prepared statement are replaced at runtime with values that you specified for a set of prepared statement parameters.

In the example, you would define two parameters:

- Define the first parameter as a **LONG** with value **1000000**. (Alternatively, you could use a field replacement to fetch the value from a variable, for example.)
- Define the second parameter as a **VARCHAR** with value ***Widget*** (Again, you have the option to use a substitution to get the value from somewhere else).

Specifying that an SQL statement is a prepared statement and defining the parameters

1. In the Test Case editor, select the step.
2. First, specify that the statement for the step is a prepared statement: In the **Database Client execute Properties > Execute Step Properties** section, check the **Use prepared statement** check box.



3. In the **Prepared Statement Parameters** properties group, define one parameter for each **?** character in the query string. For details, see the **Prepared Statement Parameters** section in “Session profile property settings for Database Client sessions” on page 929.

Session profile property settings for Database Client sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Database Client

This set of property settings tells iTest how to connect to the database.

Database Client Options

You can configure the connection in one of the following ways:

- For MySQL, SqlServer, or Oracle: Specify the **type (MySQL**, **SqlServer**, or **Oracle**), **hostname or IP address** and **IP port** of the database server, and the **database/catalog name or SID**.

  or

- For a database type other than MySQL, SqlServer, or Oracle:
  - Specify the **Java class for the custom JDBC driver**, **hostname or IP address**, and **IP port** of the database server and the **database/catalog name or SID**.

    or

  - Specify the **Java class for the custom JDBC driver** and **JDBC connection string**.

In any case, you must specify the **user ID/password** credentials for the account.

Database type	Optional. Specify the type of database server. MySQL, SqlServer, or Oracle See the “Database Client Options” section above Default: MySQL Also, see “Adding a custom third-party JDBC driver to iTest” on page 933.
Database server address/host	Optional. Specify the hostname or IP address of the database server. See “Database Client Options” on page 929.
Database server IP port number	Optional. Specify the IP port of the database server. If you specify a value for **Database type**, then you can leave this property blank to use the default port for the specified database type. See “Database Client Options” on page 929.
Database/catalog name/SID	Optional. Specify the database/catalog name or SID. To connect to Oracle Database Express Edition, set the SID as **xe**. See “Database Client Options” on page 929.
Driver class	Optional. Specify the Java class for the custom JDBC driver. For example, **com.mysql.jdbc.Driver** See “Database Client Options” on page 929. Also, see “Adding a custom third-party JDBC driver to iTest” on page 933.
JDBC connection string	Optional. Enter the URL of the JDBC connection. For example, **jdbc:mysql://[host][:port]/[database]** See “Database Client Options” on page 929. Also, see “Adding a custom third-party JDBC driver to iTest” on page 933. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
User ID	**Required**. Specify the username used to connect to the database server.
Password	**Required**. Specify the password for the **User ID** account.

Execute/Next Step

Fetch single record at a time	This property controls what is returned when a statement that return records is executed or the next page or record is fetched: each **execute** step, each click of **Fetch Additional Records**, or each **next** step. Check the box to return a single record. This is helpful for steps in loops. Uncheck the box to return a page of data each time the server returns data. If less than one page of data remains to fetch, then fetch all remaining records. **Note** If you check both **Fetch single record** and **Use first column as key**, then only **Fetch single record** is actually applied to the step. Default: unchecked
Use first column as key if unique	Check the box to identify the first column as the table key. Uncheck the box to treat the first column like any other column. **Note** If you check both **Fetch single record** and **Use first column as key**, then only **Fetch single record** is actually applied to the step. Default: checked
Records per page	Specify the number of records for the server to return each time that you request the a page of data: each **execute** step, each click of **Fetch Additional Records**, or each **next** step. Default: 100
Max cell width	In table mode, values are truncated to this number of characters. Default: 400
Use prepared statement	Check the box to cause iTest to interpret the SQL statement that you submit (either typed directly or selected from the statement templates drop-down list) to be interpreted as a prepared statement. In a prepared statement, the "**?**" characters in the statement are replaced with the values of the parameters that you specify in the **Prepared Statement Parameters** properties. This property is typically set for a particular step in a test case rather than in the session profile (for all steps in a session). For a discussion on how you might use prepared statements, see "Generating SQL statements dynamically: Using parameters in prepared statements" on page 928. Uncheck the box to interpret the statement in the normal manner. Default: unchecked

Prepared Statement Parameters

The **Prepared Statement Parameters** properties are used only if you check the **Use prepared statement** property (in the **Execute/Next Step** property group). For a discussion on how you might use prepared statements, see "Generating SQL statements dynamically: Using parameters in prepared statements" on page 928.

Note Do not confuse prepared statements with statement templates.

- **Prepared statements** are statements that include parameter references that will be replaced at runtime with the parameter values.

- **Statement templates** are general-purpose statements that you can select from a drop-down text box while performing interactive testing and then edit as needed (described in “Statement Templates” on page 933).

Adding a prepared statement parameter

1 Uncheck **Include inherited values** to allow you to add a parameter definition. (For a discussion on inheriting settings from reference session profiles — the **Include inherited values** checkbox — see “Property values: Inheriting settings” on page 111.)

2 Click  to add a new parameter definition (use  to delete a definition). Each new parameter definition appears in the list.

Data type	VARCHAR, NVARCHAR, BOOLEAN, DATE, TIME, TIMESTAMP, DOUBLE, FLOAT, SHORT, INT, LONG, URL
Value	Specify a default value for the parameter.
Date format	For **DATE**, **TIME**, and **TIMESTAMP** data types, specify the format to use. Format strings are based on Java's SimpleDateFormat. See http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html Default: **yyyy-MM-dd HH:mm:ss.SSS**

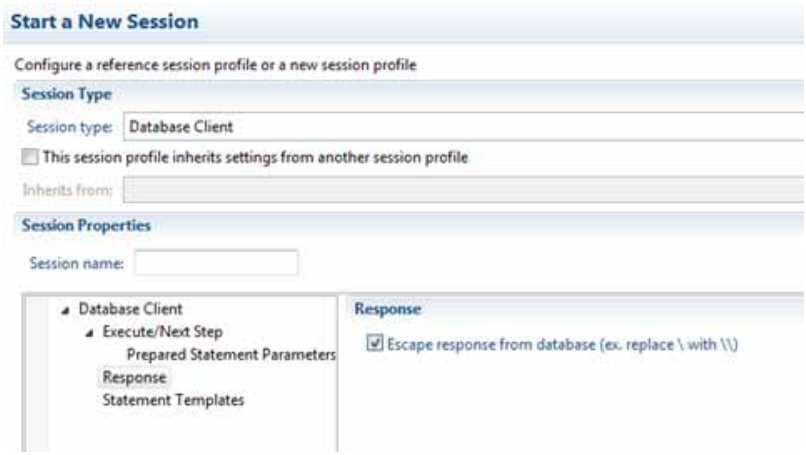
3 The order of the parameters in the list is important because the **?** characters in the prepared statement are substituted in the listed order. Re-order the list as needed using the up- and down-arrows  .

Response

The **Database Response** properties allow some characters to be escaped only when you select the **Escape response from Database** option. This option is selected by default.

If not selected, some characters will be unescaped from **Database Response**.

Note Uncheck **Escape response from Database** if you wish some characters to be unescaped from **Database Response.**



Statement Templates

Note Do not confuse statement templates with prepared statements.

- **Statement templates** are general-purpose statements that you can select from a drop-down text box while performing interactive testing and then edit as needed.
- **Prepared statements** are statements that include parameter references that will be replaced at runtime with the parameter values.

Statement templates	This property helps users of the database with predefined statements. If you specify statement templates here, then the statements appear in a drop-down text box at the top of the **SQL Statement** section of the session window. The user can edit a statement after selecting it. Specify the templates using the following format: **name:template** Separate definitions using a semicolon. For readability, we recommend that you use new lines to separate clauses. For example, in the **pets** database, you might define the following statements: display big dogs: SELECT * FROM pets WHERE type = dog AND weight > 25 ORDER BY size DESC; display all cats: SELECT * FROM pets WHERE type = cat ORDER BY breed DESC; In this example, the **display big dogs** and **display all cats** statements appear in the drop-down list. When the user selects **display big dogs**, the associated statement text appears in the **SQL Statement** text box. The user can then edit and execute the statement. Because you can use field substitutions in the statements, you can use data that is returned by other sessions when peeking or poking into the database.

Adding a custom third-party JDBC driver to iTest

Note Make sure the jdk is installed on your machine, If not, download from the following location and install it:

`http://www.oracle.com/technetwork/java/javase/downloads/jdk6u37-downloads-1859587.html`

Backup the `com.fnfr.itest.databaseservice_x.x.x.xxxx.jar` file in the `<iTest installation>\plugins directory`

1. Extract file `com.fnfr.itest.databaseservice_x.x.x.xxxx.jar` from `<iTest_installation>\plugins` to a different location, for example, `D:\temp` and update the `MANIFEST.MF` file, as follows:

a Copy the JDBC jar into the **plugins** folder (you will see other jars in the folder):

```
C:\Program Files\Spirent\iTest
<version>\plugins\com.fnfr.itest.databaseservice_<pluginVersion>
```

b Edit the **MANIFEST.MF** text file, located in:

```
C:\Program Files\Spirent\iTest
<version>\plugins\com.fnfr.itest.databaseservice_<pluginVersion>\META-
INF
```

Important In **MANIFEST.MF**, you must use a maximum line width of 71 characters.

Edit the line that starts with **Bundle-ClassPath**. Add your jar file name as the last item in the list followed by a comma. In the following example, we add **sqlitejdbc-v056.jar** just before the final `".".` (remembering the 71-character limit):

```
Bundle-ClassPath: mysql-connector-java-5.1.7-bin.jar,ojdbc6.jar,sqljdb
c.jar,postgresql-8.4-701.jdbc4.jar,sqlitejdbc-v056.jar,.
```

c Delete the **.metadata** folder of the workspace.

d Delete the **USER_HOME/.iTestConfiguration_<version>** folder.

2 Open the command prompt terminal

3 Go to the path where all the files has been extracted, for example, `D:\temp`.

```
D:\temp> C:\Program Files\Java\jdk1.6.0_37\bin
```

4 Use the command below.

```
D:\temp> C:\Program Files\Java\jdk1.6.0_37\bin\jar cvfm <jar_file_name>.jar
META-INF/MANIFEST.MF META-INF/FANFARE.DSA META-INF/FANFARE.SF com about.html
plugin.xml <jdbc_jar_file_name>
```

Example:

```
D:\temp> C:\Program Files\Java\jdk1.6.0_37\bin\jar cvfm
com.fnfr.itest.databaseservice_4.1.4.59908.jar META-INF/MANIFEST.MF
META-INF/FANFARE.DSA META-INF/FANFARE.SF com about.html plugin.xml xxxxx.jar
yyyyy.jar zzzzz.jar
```

5 This will create a new jar file in the below location with the name specified in the above command,

```
D:\temp
```

6 Copy and paste this new jar file in the iTest installation plugins directory. (Overwrite the existing file when it prompts, since we have already taken back up).

Note The jar file name `<jar_file_name>.jar` should be the same as the default jar file name which was extracted in Step 1. Also, as mentioned earlier, make sure that you have a backup of the default jar file before replacing it with the new jar file.

7 Delete the `.metadata` folder in the workspace.

8 Restart the iTest via command prompt using the –clean option,

```
C:\Program Files\Spirent Communications\iTest 4.2>iTest.exe –clean
```

9 Specify the appropriate session profile property settings in Database session to connect third-party JDBC database. In this example the following is used:

Database type: Other

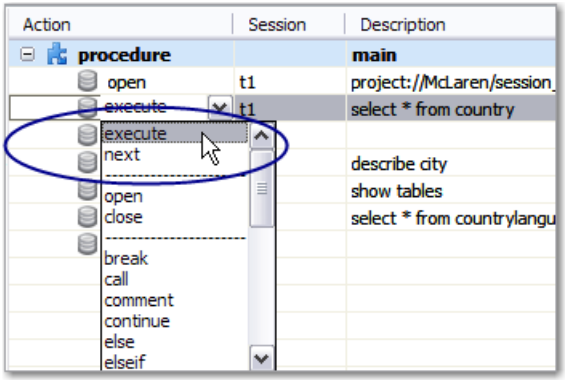
Driver class: org.sqlite.JDBC

JDBC connection string: jdbc:sqlite:vqa.db

Database Client command set

This topic describes all Database Client commands that you can submit from iTest.

1 For a step in an Database Client session, select one of the listed actions in the **Action** cell.



2 If applicable, type the SQL statement in the **Description** cell. Field replacements are supported in statements.

Command reference

execute	The **execute** command executes the SQL statement that you specify in the **Command** property (**Description** cell) for the step. For statements that return records, the command returns the first record or first page of records, depending on the setting for the **Fetch single record at a time** property in the session profile. In an interactive session, when you click **Execute**, iTest captures an **execute** action
next	The **next** command fetches the next page of records or the next record, depending on the setting for the **Fetch single record at a time** property in the session profile. To fetch the first record or first page of records, perform the **execute** command. You can specify the number of records per page using the **Records per page** property in the session profile. In an interactive session, when you click **Fetch Additional Records**, iTest captures a **next** action.

CHAPTER 44

File sessions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

About File sessions

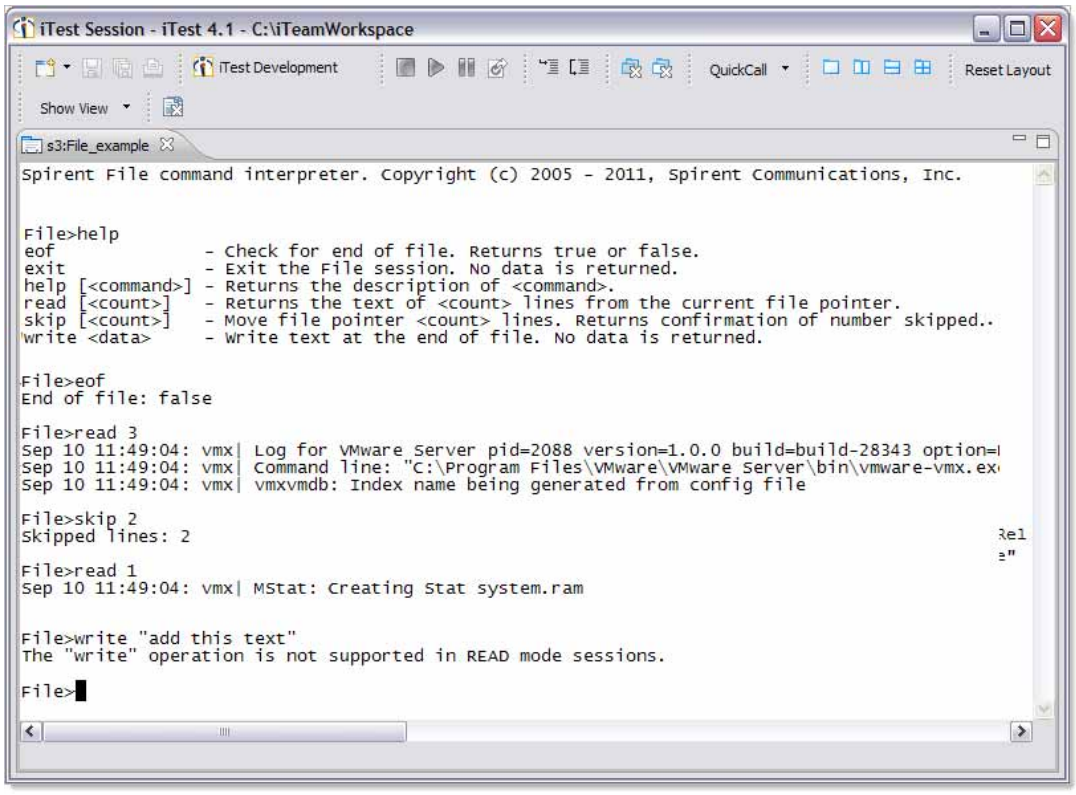
The File session type enables an automated test case to work with text files during execution (open a file, go to a specified position in the file, read a line, write a line, and so on).

You can use the interactive File session editor to execute commands that you can save as steps in an automated test case.

Note A File session can open a file for either reading or writing, but not both in the same session.

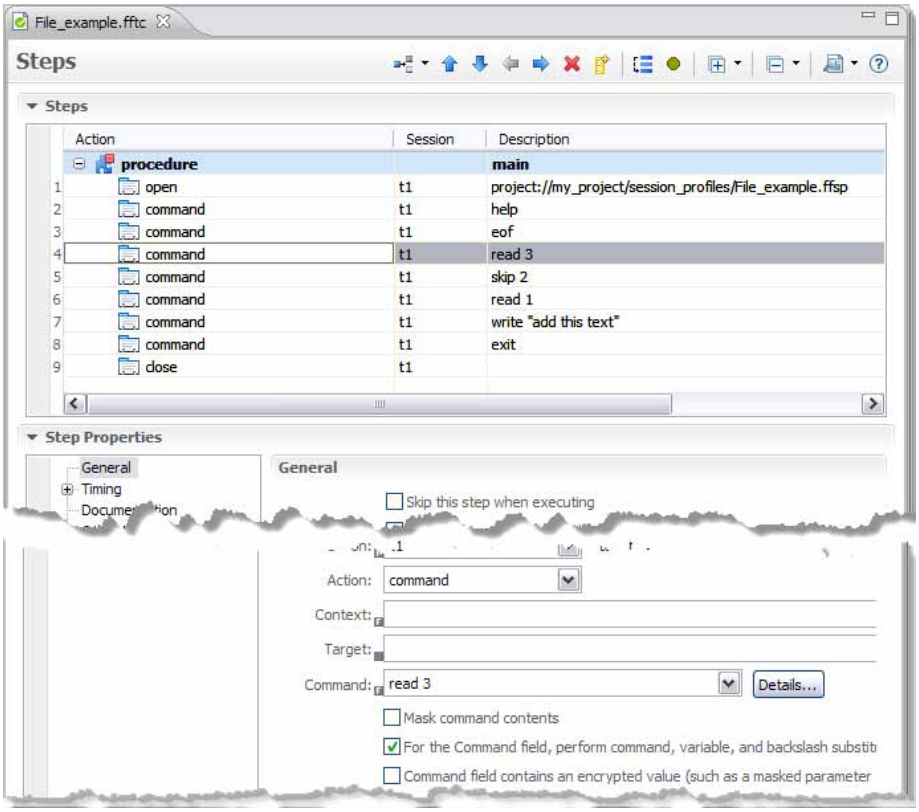
Example File session

In this example File session in iTest, we use several of the available commands.



File test cases

Here is the test case that results when we save the captured steps (notice that we performed an **exit** step after the steps in the example):



Modifying command properties at runtime

- In test cases, all File step properties support field replacement (properties with the F indicator), so you can dynamically control a step at runtime. For example, you can dynamically set the number of lines to read in a **read** step using a variable. Let's say that step 16 set the value of the variable named `linesToRead`. Step 17 could use the **command** text `read $linesToRead`

  See Chapter 28, "Field Replacements".

- You can use also field replacements to provide values for most properties in File session profiles. For example, for the **URI** property, you could use: `[param file_uri]`
- For the **open** step, you have the option to override any of the property settings so that all steps in the session use the new property settings. Change any of the properties for the **open** step in the **<sessionType> Session Properties** section. See "Step Properties section: Session Properties: Overriding testbed device or session profile settings in the open step" on page 183.

File session command reference

File session steps support the following commands:.

eof	Returns **true** if the file pointer is at the end of file. Otherwise returns **false**. **Tip** Store the response in a variable for use as a loop controller while reading individual lines in a loop.
exit	Closes the file and closes the FIle session.
help [*command*] Alternatively, type **?**	Returns the description of the specified *command*. If no *command* is specified or if **?** is used, returns the full list of commands and descriptions.
read *lineCount*	Returns the specified number of lines from the file. The file pointer remains at the beginning of the line after the last read line. **Note** The **Access mode** property must be set to **READ**.
skip *lineCount*	Starting at the current file pointer position, moves the pointer the specified number of lines.
write *dataToWrite*	Writes the *dataToWrite* text at eof (after the last line). If the text includes space characters, enclose the text in quotes.The **Access mode** property must be set to **WRITE**. **Note** **write** mode is not supported for **zip**, **jar**, **tar**, **tgz**, or **tbz2** file types. **write** mode is not supported when the URI uses **HTTP**, **HTTPS**, or **SFTP**. **Tip** Use substitution to specify the text dynamically.

Session profile property settings for File sessions

File sessions open the file at the specified **URI** using either of the specified **Access mode** — **read** or **write**. You cannot read and write in the same session.

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click

.

Session properties, File

URI	Specify the URI that locates the file. **Supported notation:** **Local Files:** project:// project-path [file://] absolute-path **Zip, Jar and Tar:** zip:// arch-file-uri[! absolute-path] jar:// arch-file-uri[! absolute-path] tar:// arch-file-uri[! absolute-path] **gzip and bzip2:** tgz:// arch-file-uri[! absolute-path] tbz2:// arch-file-uri[! absolute-path] **HTTP, HTTPS, FTP, FTPS and SFTP:** htttp://[username[: password]@] hostname[: port][absolute-path] htttps://[username[: password]@] hostname[: port][absolute-path] ftp://[username[: password]@] hostname[: port][absolute-path] ftps://[username[: password]@] hostname[: port][absolute-path] sftp://[username[: password]@] hostname[: port][absolute-path]
Encoding	Specify the encoding type for the text file. Default: UTF-8
Access mode	Specify how to work with the file. **read** text from the file **write** text into the file. **Note** **write** mode is not supported for **zip**, **jar**, **tar**, **tgz**, or **tbz2** file types. **write** mode is not supported when the URI uses **HTTP**, **HTTPS**, or **SFTP**.

File > Authentication

Username	Optional. Specify the username used to connect to access the file.
Password	Optional. Specify the password. The text is masked here and in all locations where it is used.

File > Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

CHAPTER 47

HP ALM (formerly HP Quality Center Server)

How iTest and HP ALM work together

Note HP ALM is supported on Microsoft Windows only.

ALM features require a separate license.

iTest is strongly integrated with ALM. You can start test execution and monitor the results in both applications.

- In the ALM **Test Plan** page, you can create a test of type **ITEST** or you can convert an existing test to **ITEST** type.
- From the ALM **Test Script** page, you can start iTest and edit the test case. (This is possible because QC can maintain project/path information for the associated iTest test case.)
- You can define parameters and set their values in either QC or iTest
- You can associate any iTest test case step with a design step in the associated ALM test.
- iTest can publish design step information and other information to the specified ALM test. iTest can auto-generate and auto-name new QC steps when you publish a test case. You can publish either one test at a time or multiple tests as a batch.
- iTest can publish test report data to the specified ALM test instance. You can publish either one test report at a time or multiple reports as a batch.
- itestcli can automatically publish test reports when test case execution finishes

Overviews of the various uses of iTest and QC follow. You will find detailed instructions later in this chapter.

Specifying the workspace to use for ALM files

You can specify the workspace that iTest should use when working on ALM tests using one of the following options:

- You can use the workspace that is specified in the **HPQC_ITEST_WORKSPACE** system environment variable.
- If there is no workspace specified by the environment variable, then iTest uses the **iTest_HPQC_Workspace** folder in your home directory.

Note For more installation and configuration information, see the *iTest for ALM Installation Guide*.

Overview: Connecting iTest and ALM tests

Typical workflow for integrating ALM and iTest

1. Write up the test in ALM. Test planning is performed by detailing the ALM design steps.
2. Based on the plan, perform a quick dry run to make sure that all scenarios are well thought out.
3. If needed, make changes to the design steps in ALM and do the dry run again.
4. Repeat the process until the test plan script works as required.
5. Start capturing in iTest, and perform the entire test manually.
6. Stop capturing and create a iTest test case from the captured steps.
7. Clean up the iTest test case by removing unwanted steps and typos.
8. Execute the test case in iTest until it executed as needed.
9. Add pass/fail analysis rules and control steps (loops and other logic).
10. Run the test case in iTest several times and make sure the test case passes. Modify the test case as needed.
11. Once the test case passes, publish both the test case and the test report to ALM

Working with ALM tests

To work with ALM tests, you have the following options:

Start in ALM Start with a new or existing test of type **ITEST** in QC, add design steps and define custom properties as needed. Then click a button to open the iTest Test Case editor so you can create the associated test case in iTest.

Start in iTest Work on a new or existing executable iTest test case and then publish it to a new or existing associated test in QC.

Let’s look at the options:

Starting in ALM

1. In the **Test Plan**, start with a new test of type **ITEST** or select an existing test and set its type to **ITEST**. Create design steps as needed.
2. When the test is ready, on the **Test Script** tab, click **Edit Using iTest**. iTest starts and asks you to specify the location of the new iTest test case file (by default, it uses the same name as the associated QC test).
3. Now you start work in iTest. (Meanwhile, in QC, you can now navigate away from the **Test Script** tab to work on other pages as needed.)
   - iTest creates the test case and uses **Comment** steps to correspond to the QC design steps
   - Ensure that you have the latest shared test assets from version control

- In iTest, populate the test case steps (typically by capturing interactive sessions). Organize the sessions to be nested under the **Comment** steps.

  For each additional step that you want to associate with a QC design step, you specify **Quality Center** property settings (for example, the name of the QC step and the expected result). You have the option to have iTest auto-update step names when publishing to QC or to use the step names as they currently exist in QC.

- Add analysis rules as needed
- Define parameters and insert them into test case steps and properties. See "Overview: Parameters in iTest and ALM" on page 962.
- Debug the test case

4 Run the **Publish Test Case to ALM** wizard to publish the updated test case to the associated test in QC. The wizard helps you to configure QC server/login/domain information (you can have the wizard remember the settings) and lets you specify how to update design steps in the ALM test.

5 Optional, to enable you to publish test reports: After you execute the test case, you can run the **Publish Test Report to ALM** wizard to associate the test report with one or more test instances in one or more test sets in QC.

- You can upload two types of iTest data to ALM server: execution results for the test instance and iTest test reports as HTML-format files.
- You have the option to publish reports on demand or to have iTest auto-publish when you exit iTest.
- When executing iTest test cases using itestcli, you have the option to auto-publish test reports to QC
- If you execute iTest test cases using itestcli, you can use the **-qc** command line option to auto-update the appropriate ALM test instances with iTest test report data when test execution finishes.

  ```
  Check the test case in to version control.
  ```

Starting in iTest

1 In iTest, load the latest shared test assets from version control.

If you need to perform other ALM tasks, click **Open ALM** in the iTest Test Case editor to open a browser pointing to the ALM server.

2 Create a new test case using the iTest **New Test Case** wizard.

- **Existing QC test:** On the appropriate wizard page, associate the test case with an existing QC test. As a result, iTest populates the outline of the test using top-level **Comment** steps corresponding to the design steps.
- **New QC test:** Add a new QC test and/or test instance.

3 In iTest's Test Case editor, populate the test case steps (typically by capturing interactive sessions):

- Nest sessions or particular steps under the **Comment** steps
- For each additional step that you want to associate with a QC design step, you specify **Quality Center** property settings (for example, the name of the QC step and the expected

result). You have the option to have iTest auto-update step names when publishing to QC or to use the step names as they currently exist in QC.

- Add analysis rules as needed
- Define parameters and insert them into test case steps and properties. See "Overview: Parameters in iTest and ALM" on page 962.
- Optional: Associate reference files with the test case if desired. See "Associating reference files with a test" on page 961.
- Debug the test case

4 Next, run the **Publish Test Case to ALM** wizard to publish the updated test case to the associated test in QC. The wizard helps you to configure QC server/login/domain information (you can have the wizard remember the settings) and lets you specify how to update design steps in the ALM test.

5 Optional, to enable you to publish test reports: After you execute the test case, you can run the **Publish Test Report to ALM** wizard to associate the test report with one or more test instances in one or more test sets in QC.

- You can upload two types of iTest data to ALM server: execution results for the test instance and iTest test reports as HTML-format files.
- You have the option to publish reports on demand or to have iTest auto-publish after any test case execution. If you execute a test from iTest, you must publish the test report manually. If you execute a test from QC, the test report is auto-published.
- If you execute iTest test cases using itestcli, you can use the **-qc** command line option to auto-update the appropriate ALM test instances with iTest test report data when test execution finishes.

6 Check the test case in to version control.

Overview: Executing ALM test sets

Because you can view iTest and QC simultaneously, you might follow this sequence to execute:

1 In the QC Test Lab, select a test set.

2 If needed, update parameter settings and then execute the test set. See "Overview: Parameters in iTest and ALM" on page 962.

3 Monitor execution (do not close the browser session with QC).

4 When execution completes, move to QC and ...

- Review the iTest HTML-format test report that has been published into QC
- Review the test instance for the run
- Review any additional artifacts that have been published (for example, log files)

Creating a new test in ALM and editing it using iTest

1 In the ALM **Test Plan** pages, click **New Test.** In the **Create Test Case** dialog box, specify the following settings:

Test Type	Select **ITEST** from the drop-down list. This results in a iTest type test that is stored by ALM. You will use iTest to edit it and ALM to run it.
Test Name	Type the name of the test. iTest will use this name when opening the test case in the Test Case editor.
Template	Not supported.

2 Click the **Test Script** tab. The page displays an empty test. Click **Edit Using iTest.** iTest starts the **New Test Case** wizard.

New Test Case page

3 Specify the **Container** (the parent folder). When you click **Browse,** the explorer box displays all projects in the current workspace and selects a recommended folder. You can navigate to any project or folder.

4 Type a name for the new document in the **File name** text box.

5 Click **Finish.** If iTest is not configured to save your QC login credentials, you have one additional page; click **Next.**

Login page

6 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next.**

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.

Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Test Instance page

7 The **Test Instance** page displays the tree from the ALM **Test Lab** page. Select the appropriate folder, select the appropriate test set, and then select the test instance to update.

Note If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Instance page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Add Test Instance	Click to add a test instance to the selected test set.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

8 Click **Finish**.

9 The iTest Test Case editor opens the new test case.

- Add steps either manually or by capturing manual sessions.
- For steps that should be associated with design steps in ALM, specify appropriate property settings. See "Editing step properties: The Quality Center properties group in the Step Properties section" on page 960 for details.
- Define and populate parameters as needed. See "Overview: Parameters in iTest and ALM" on page 962.
- Optional: Associate reference files with the test case. See "Associating reference files with a test" on page 961.

10 When you finish editing the test case, execute it and make any required changes.

11 When ready, save the test case.

12 Publish the iTest test case to QC.

13 The test now appears on the **Test Script** page. Click the **Design Steps** tab to view the new design steps.

Creating a new iTest test case that will be associated with ALM test

While using the **New Test Case** wizard to create a new test case document, if you select **Associate this test case with a test in ALM**, then the wizard presents additional pages where you configure the connection to the QC server and specify the QC test and test instance to associate with the test case.

1 Create a new iTest test case document using one of the following methods:

- In the iTest Explorer, right-click the intended parent folder and select **New > Test Case**.
- In the main menu, click **File > New > Test Case**.
- In the main toolbar, click **New Document**  and select **Test Case**.

New Test Case page

2 The **New Test Case** wizard starts. Specify the **Container** (the parent folder). WHen you click **Browse**, the explorer box displays all projects in the current workspace and selects a recommended folder. You can navigate to any project or folder.

3 Type a name for the new document in the **File name** text box.

4 When you select **Associate this test case with a test in ALM**, then the wizard presents additional pages where you configure the connection to the QC server and specify the QC test and test instance to associate with the test case. Click **Next**.

Login page

5 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next**.

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.

Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Domain and Project page

6 On the **Domain and Project** page, specify the following information for the test and then click **Next.**

Domain	Specify the domain of the ALM server. **Note** To pre-populate the setting, you can specify the default **domain.** (Click **Window > Preferences> Spirent > Quality Center** - "ALM preference settings" on page 988)
Project	Specify the project.

Test page

7 The **Test** page displays the tree from the ALM **Test Plan** page. Select the appropriate folder and then select the test.

Note If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Plan Folder	Click to add a test plan folder under the selected folder.
Add Test	Click to add a test under the selected folder.
Delete	In the ALM file structure, select a test plan folder or test and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test plan folder or test and then click **Edit** to modify its **Name** or **Description** text. For tests, you can also modify the **Status** and **Designer** settings. **Status**: Specify **Ready**, **Design**, **Import**, or **Repair**. Default: Ready **Designer**: Select the appropriate designer.
refresh	If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Test Instance page

8 The **Test Instance** page displays the ALM **Test Lab** tree view.

To cause itestcli to automatically publish test report data, you must associate the test case with a ALM test instance.

Select the appropriate folder, select the appropriate test set, and then select the test instance to update.

To set the test instance that the iTest test case is associated with, check **Associate the iTest test case with the selected ALM test instance** and then select the test instance in the tree view.

Note If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Instance page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Add Test Instance	Click to add a test instance to the selected test set.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click Delete to delete it.
Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Click **Finish**. Now, whenever you make changes to a step that is associated with a ALM test and exit iTest or switch workspaces without having published the test case, iTest asks whether to publish the changes to the ALM test.

Working on an existing test in ALM and editing it using iTest

While working on a test on the **Test Script** page in QC:

- For an existing test that has not yet been associated with a iTest test case, be sure to set the **Test Type** to **ITEST**.
- When you click **Edit Using iTest**, a wizard presents the **Login** page to so that iTest can learn your login credentials.

1 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center**

page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next**.

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.
Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

qc command: Getting and setting the location of the Artifacts folder

The **qc** command applies only if you are using iTest for ALM.

Syntax

qc setArtifactsLocation *URI*

Description

Sets the **artifactsLocation** property in the **QualityCenterInfo** section of the test report. When execution finishes, iTest zips the artifacts and saves the zipped file to the specified location.

Note You can also set the value using the **Artifacts location** property on the **Quality Center** page in the Test Case editor.

Syntax

qc getArtifactsLocation [*defaultURI*]

Description

Returns the value of the **artifactsLocation** property in the **QualityCenterInfo** section of the test report. If not found, returns the default URI that you have the option to specify using [*defaultURI*].

Response on success

Text response: URI of the Artifacts folder.

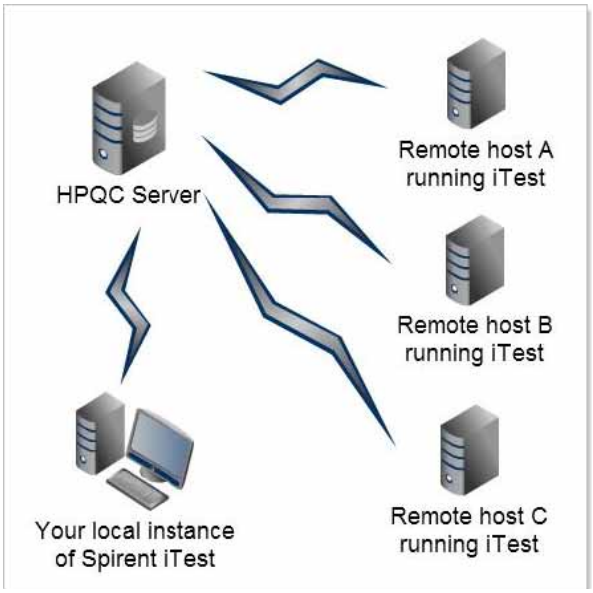
Running a iTest test case from ALM

1. First, create a test of type **ITEST**, as described in "Creating a new test in ALM and editing it using iTest" on page 948.
2. On the **Test Lab** pages, create a test set that includes the test that you created.
3. Run the test as you would run any ALM test. On the **Execution Grid** page of the Test Lab, you can select any **Planned Host Name** from the list.

   itestcli (the command-line executable version of iTest) executes the test. You will see the Windows Command Prompt application open and execute itestcli with appropriate options and arguments.
4. When execution is complete, close the **Automatic Runner** window.
5. iTest publishes the test report to ALM in HTML format. You can view the report in the **Test Instance Properties** window.

Executing iTest test cases on remote hosts

Typically, you execute test cases locally — the instance of iTest that executes the test cases and the instance of Internet Explorer that is accessing the HPQC server are on the same computer. Alternatively, you can execute remotely — specify that tests cases should execute on remote hosts that are running instances of both iTest and IE accessing HPQC. The remote hosts run the test cases using itestcli.

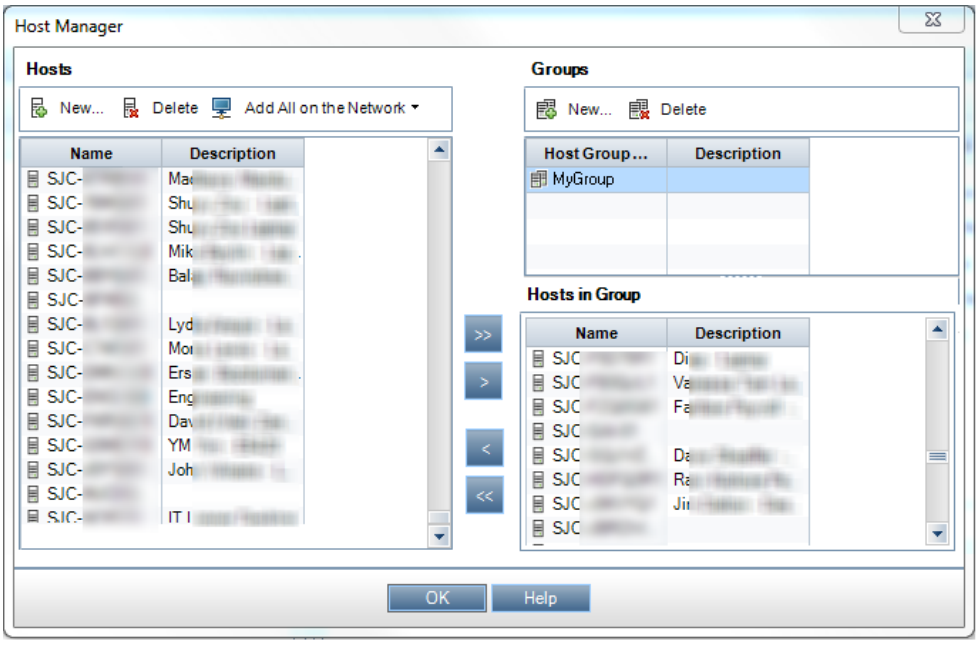


Configuring remote execution

Step 1 Specify the remote hosts that will execute test cases

1. Log into HPQC and go to Test Lab.

2 On the **Test Sets** tab, click **Test Sets > Host Manager**. On the **Host Manager** dialog, add the remote host to a group.



Step 2 On each remote host: Configure the client system environment

1 Add the following environment variables under **System Variables**:

Add the iTest installation directory to the **path** variable

Add **HPQC_ITEST_WORKSPACE** and set it to **C:\<path to iTest workspace>**

Add **HOMEDRIVE** and set it to **C:**

XP: Add **HOMEPATH** and set it to **\Documents and Settings\<userName**, for example, **Administrator>**

Windows 7: Add **HOMEPATH** and set it to **\Users\<userName**, for example, **Administrator>**

2 Install QC add-ins:

a For Windows 7 and Windows 2008 Server, open Internet Explorer as administrator.

b Go to: **http://<QC_Server domain>:8080/qcbin**

c Click the **Add-ins** link

d Install the **HP Quality Center Connectivity** Add-in

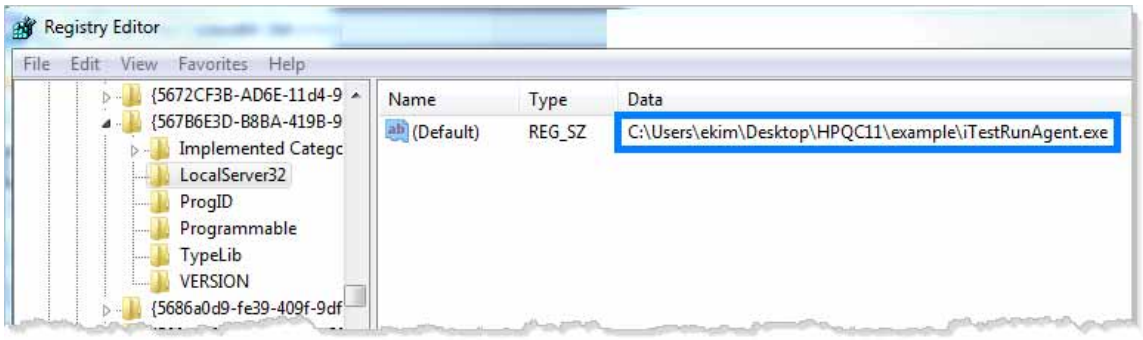
e Install the **HP ALM Client Registration** Add-in

3 If Windows Firewall is enabled, add **iTestRunAgent.exe** as an exception (**iTestRunAgent** launches the **cmd** process for **itestcli.bat**).

To find the **iTestRunAgent** path in your windows registry:

a Click **Start > Run** and run **regedit**

b Search the registry for **iTestRunAgent** to find the path of the executable that is registered.



Step 3 On each remote host: Modify the DCOM security properties

1 Click **Start > Run** and run **dcomcnfg**

2 Navigate to **Console Root > Component Services > Computers > My Computer**

a Right-click **My Computer** and select **Properties**

b On the **Default Properties** tab, make sure that the **Default Impersonation Level** is **Identify**

c On the **COM Security** tab.

For **Access Permissions**: Click **Edit Default** and then select **Local Access** and **Remote Access** permissions for all users and groups. Repeat for **Edit Limits**.

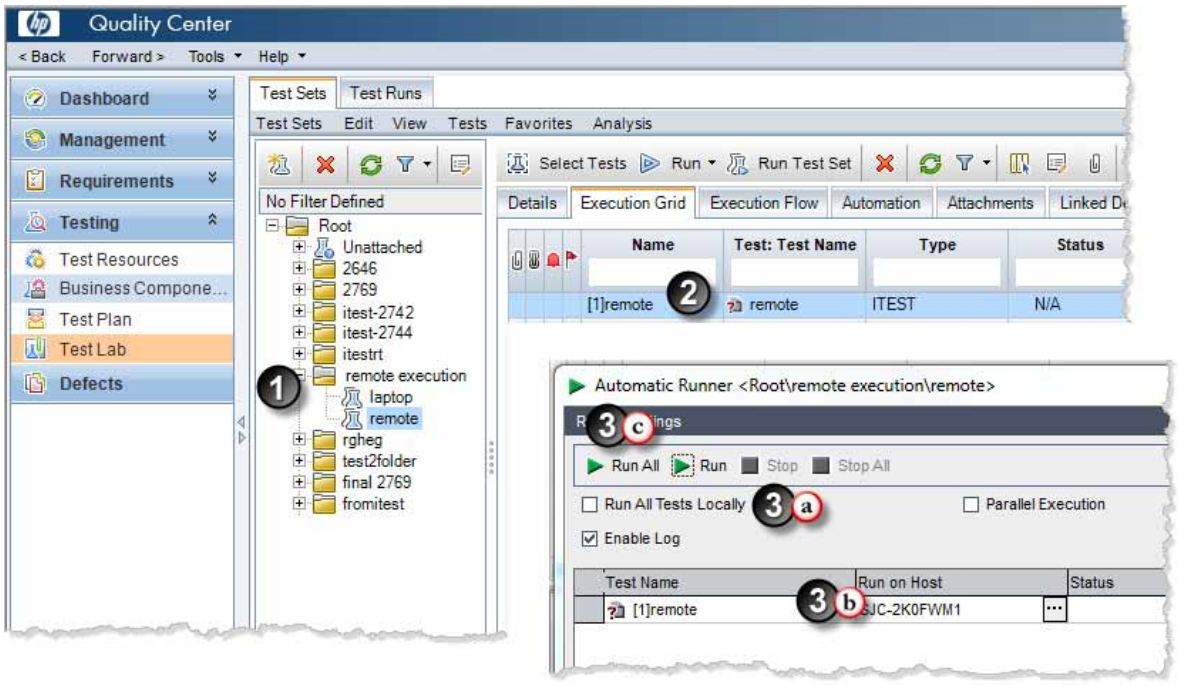
For **Launch and Activation Permissions**: Click **Edit Default** and then select **Local Launch**, **Remote Launch**, **Local Activation**, and **Remote Activation** permissions for all users and groups. Repeat for **Edit Limits**.

3 Restart the remote host.

To execute a test case on a remote host

1 Log in to QC and select the test set to execute.

2 On the **Execution Grid** tab, select a test and click **Run**. Alternatively, click **Run Test Set** to run all tests in the test set.



3 On the **Automatic Runner** page:

a Uncheck **Run All Tests Locally**

b Specify the remote host in the **Run on Host** column for each test case

c Click **Run All** to run all the test cases, or select a single test and click **Run**

Test Case editor: Quality Center page

Use the **Quality Center** page to perform the following operations:

- **Start a session with the ALM server**

  Start a session with the ALM server (for example, to enter additional information about the test). To start a session, click **Open Quality Center**.

- **Run the Publish Test Case to Quality Center wizard**

  - Run the **Publish Test Case to Quality Center** wizard to configure QC server information and to update the appropriate design steps (and other information such as **Owner**, **Description** parameters, step names, and so on) in the associated ALM test.

    Once you run the wizard, the **Quality Center Server Information** section displays the current settings and the QC test is updated whenever you save the iTest test case. Click **Publish to Quality Center** to start the wizard.

    Note If you plan to execute a test case using itestcli, then, before executing the test case, run the wizard to publish the test case to ALM. This action sets the domain, project, and

test instance location for the test case so that itestcli can auto-update the test instance with iTest test report data.

Quality Center Server Information

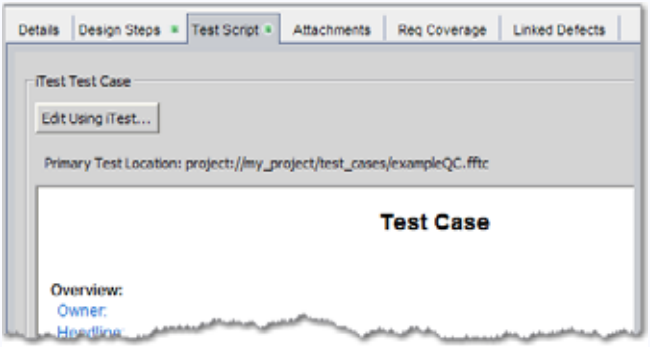
Domain	Displays the name of the ALM domain in which the associated ALM test case resides.
Project	Displays the name of the ALM project within the specified domain in which the test resides.
Test plan location	Required. Displays the path from the root of the QC project to the associated test in the ALM Test Plan:
Test instance location	Displays the test instance that test results will be published to.

When you click **Publish to Quality Center**, if any of the "**Quality Center Server Information**" (above) is missing, then the information is retrieved from the stored data on the Quality Center. That is, the information you specified as described in section "Login page" on page 948, if you have selected the appropriate Preferences options (**Windows > Preferences > Spirent > Quality Center**, see "Publish the test report to Quality Center" on page 989).

- **Configure settings that affect how test cases and test reports are published to QC**

The following property settings appear in the **Publishing Behavior** section of the page:

Publishing Behavior

Artifacts folder location	Some test cases may produce artifacts (like log files) during execution. If you specify an **Artifacts folder location**, then when iTest publishes the test report to QC, the contents of the specified folder are uploaded to QC as attachments to the report. Click the link to view the folder. Default: [none]
Delete all files in the artifacts folder after publishing	Check the box to cause iTest to delete the contents of the artifacts folder after successfully publishing the contents of the folder to QC. Default: unchecked
Store primary test case information in Quality Center	Check the box to cause iTest to publish the test case location information to QC whenever you publish the test case. As a result, you will be able to open the test case in the iTest Test Case editor by clicking **Edit Using iTest** on the QC **Test Plan** page. The information appears at the top of the **Test Script** page.  If you uncheck the box, then iTest will not publish the information to QC. (If the information is currently stored in QC, then it will be deleted when you publish a iTest test case to QC.) For further discussion, see "Storing test case information in ALM" on page 970. Default: unchecked
On publishing, replace Quality Center parameters with iTest parameters	**Note** This property applies only to QC 10.0 and later. Check the box to cause the parameters specified in the iTest test case to be written to the test case published in QC. If publishing to an existing test case in QC, existing parameters will be replaced. Further information on how parameters are published appears in "Overview: Parameters in iTest and ALM" on page 962.

Design Step Renaming	
Automatically rename design steps when publishing	Check the box to cause iTest to auto-generate step names for the QC test. You can specify one of the following formats for the step names: • **Step ID**: Use the format <procedure name>:<step ID>, for example, **main:3.2.1** • **Prefix text**: Specify text that will be followed by a space character and then the step number. For example, if you specify **STEP**, then the resulting names are **STEP 1**, **STEP 2**, **STEP 3**, and so on. Default: unchecked **Note** If you check the box, then iTest overwrites the **Step name** setting that you specify for a step. See "Quality Center properties" on page 960.

Editing step properties: The Quality Center properties group in the Step Properties section

In the Test Case editor, select a step. In the **Step Properties** section, select **Quality Center**. The property settings on this page apply only if you want to associate this step with a design step in an ALM test.

If you check **Associate this step with a design step**, the data configured on this page is published to ALM. In addition, if you publish a iTest test report to ALM, then execution messages in the test report that are associated with the executed step are published to the appropriate test instance.

Quality Center properties

Associate this step with a design step	Check the box for any iTest step that should create or update the ALM design step that is specified by the **Step name** property. If you check the box, then you must specify the associated ALM design step for the **Step name** property. Default: unchecked
Step name	Specify the name of the ALM design step to associate the current iTest step with. You can specify an existing design step or a new step name that will be added to the ALM test when you publish the test to QC. We recommend that you use the ALM naming convention of **step <n>**, where **<n>** is the index number of the step. Do not start the name with a space character. Step names must be unique within a test case. **Note** iTest overwrites the **Step name** if you configure the **Automatically rename design steps when publishing** property on the **ALM** page in the Test Case editor. See "Publishing Behavior" on page 959.
Description	Optional. Provide a single line of text to document the step. This information is added to the definition of the ALM design step specified by the **Design step description** property.
Expected result	Optional. Type a simple description of the expected result for the step. This information is added to the definition of the ALM design step specified by the **Design step expected result** property.

Include execution issues from nested steps when reporting	Check the box to cause iTest to include (in the test run result for the test instance) messages for any test case steps that are nested under the current step. **Tip**: Use this setting for a top-level step (for example, an EXEC **comment** step) that documents the associated ALM design step. Nest steps under the top-level step. Then, after execution, the test results for the nested steps are all associated with the appropriate design step. Default: checked
Set Quality Center Status to Pass for steps with no analysis rules	In most tests, many design steps are not "interesting" from the testing standpoint (for example, open and close steps). For such steps, the iTest step does not include an analysis rule because there is no response to analyze. Checking the box ensures that, for any iTest step that does not have an analysis rule, the execution **Status** of the associated design step in ALM is set to **Pass**. If you uncheck the box, then, when a iTest step does not have an analysis rule, the associated design step's **Status** is set to **No Run**. **Important:** If a step fails during execution, then this setting is overridden and the execution **Status** of the associated design step in QC is set to **Fail**. In most cases, leave this setting in the default checked state. Default: checked

Associating reference files with a test

To fully define a test and to describe it for coworkers, you might want to associate files with the test (a config file and a topology diagram, for example). Use the **Reference Files** page on the Test Case editor to add links to files that should be associated with the test case. (You can also delete and update the list.)

When you publish the iTest test case to ALM, all reference files are listed on the **Attachment** page of the test in the test plan. The names appear as **attachmentName_fileName**

Important Reference files are associated with the test as supporting documentation and are not used in any way during execution.

See "Test Case editor: Reference Files page" on page 237.

Steps Report Saved to Report Database

The iTest executions steps report saved to the Report database depends on the settings of the following three parameters:

- **save(parent)**: How the parent step was saved
- **include(step)**: State of **Include this step and its children in test reports** (whether the parameter is selected or not—**Step Properties > General> Include this step and its children in test reports**)
- **associate(step)**: State of **Associate this step with a design step** (whether the parameter is selected or not—**Step Properties > Quality Center > Associate this step with a design step**)

The table below illustrates how iTest saves the **Step reports**.

save(parent)	include(step)	associate(step)	save(step)
Full	Yes (Selected)	Any	Full
Full	No (not selected)	Yes (Selected)	Short
Full	No (not selected)	No (not selected)	None
Short	Any	Yes (Selected)	Short
Short	Any	No (not selected)	None
None	Any	Any	None

Overview: Parameters in iTest and ALM

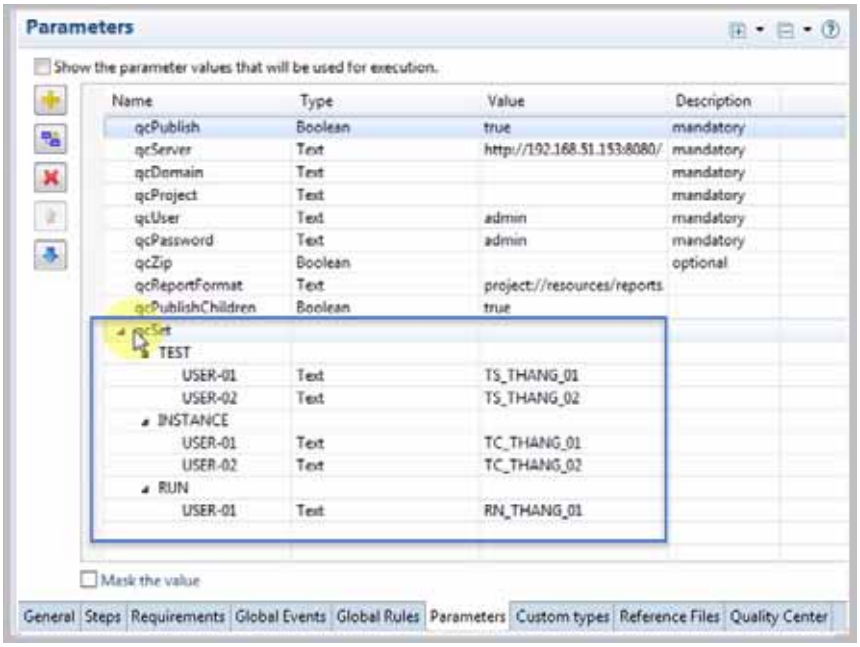
When you publish a test case to QC, iTest publishes the parameters as they would be used at runtime — parameters defined in the test case, in the associated testbed (if specified), and in the parameter file (if specified) — all after inheritance/merging.

Important When you publish a test case, parameter **name** and **description** are published to QC as defined in iTest. In contrast, parameter **values** are published as [blank] (for the reasons described in "Setting parameter values for test runs" on page 969).

Automatically populate the customization fields for instances

iTest allows you to define custom parameters for setting custom fields on an iTest HPQC Test Case that may be published and run again on HPQC. The parameters defined in iTest Test Case automatically populates the HP ALM custom filed elements: **Test, Test Instance,** and **Test Run**.

In iTest, on the **Test Case editor >Parameters tab**, define customized field for 3 elements that are automatically populated on HPALM test instance and test run. Set the three customized field element of **qcSet** as shown below:



Important The HP ALM database, reserves many fields, which may be defined as custom field when required. For example, the RN_USER_01 is a custom label which you can set as desired when creating/adding a custom field to the RUN. HP ALM will auto-map with a reserved field in the database when you create/add a custom field. In above, RN_USER_01 is mapped with USER-01 in the HP ALM database. For ease of reference, if you want to set the field as RN_USER_01 in the HP ALM database, set USER-01 in the iTest parameter file.

The following screenshots are include for your reference to show how the parameter file defined are mapped and appear in HP ALM.

Note Ensure that you publish the iTest Test Case to HP ALM.

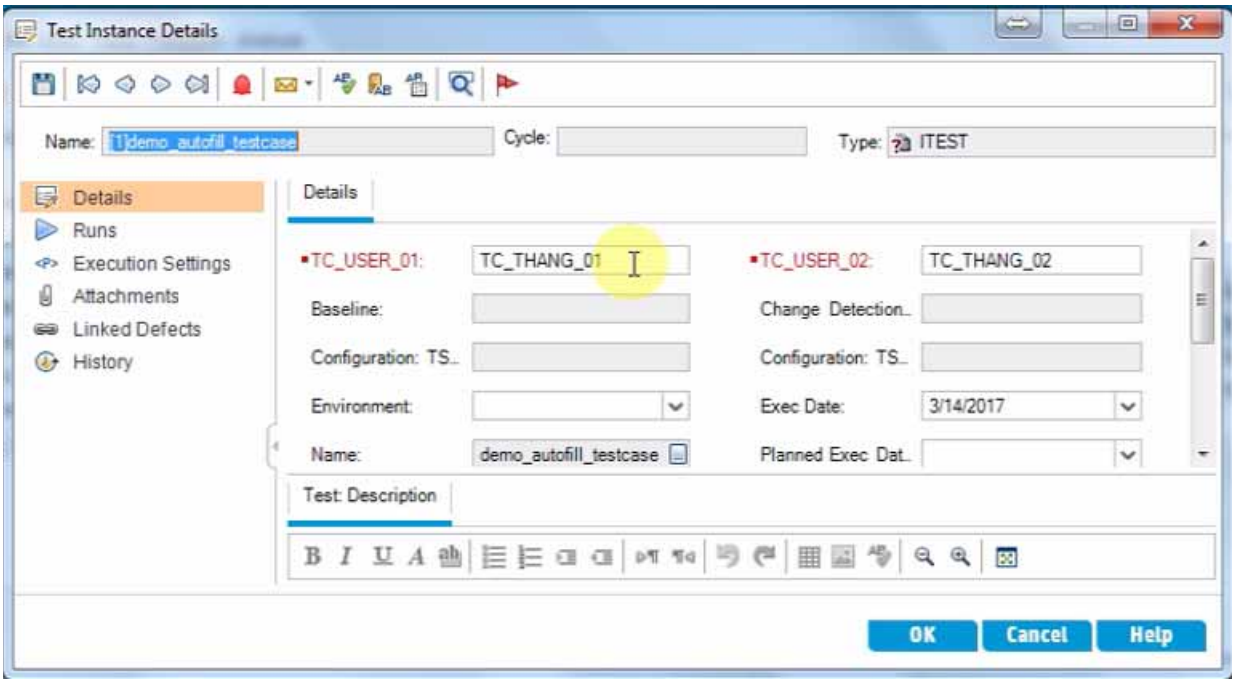
Note The precedence of parameter file vs the Test Case defined parameters used with the **qcSet** option is as follows:

iTest GUI: Param file > Test case param

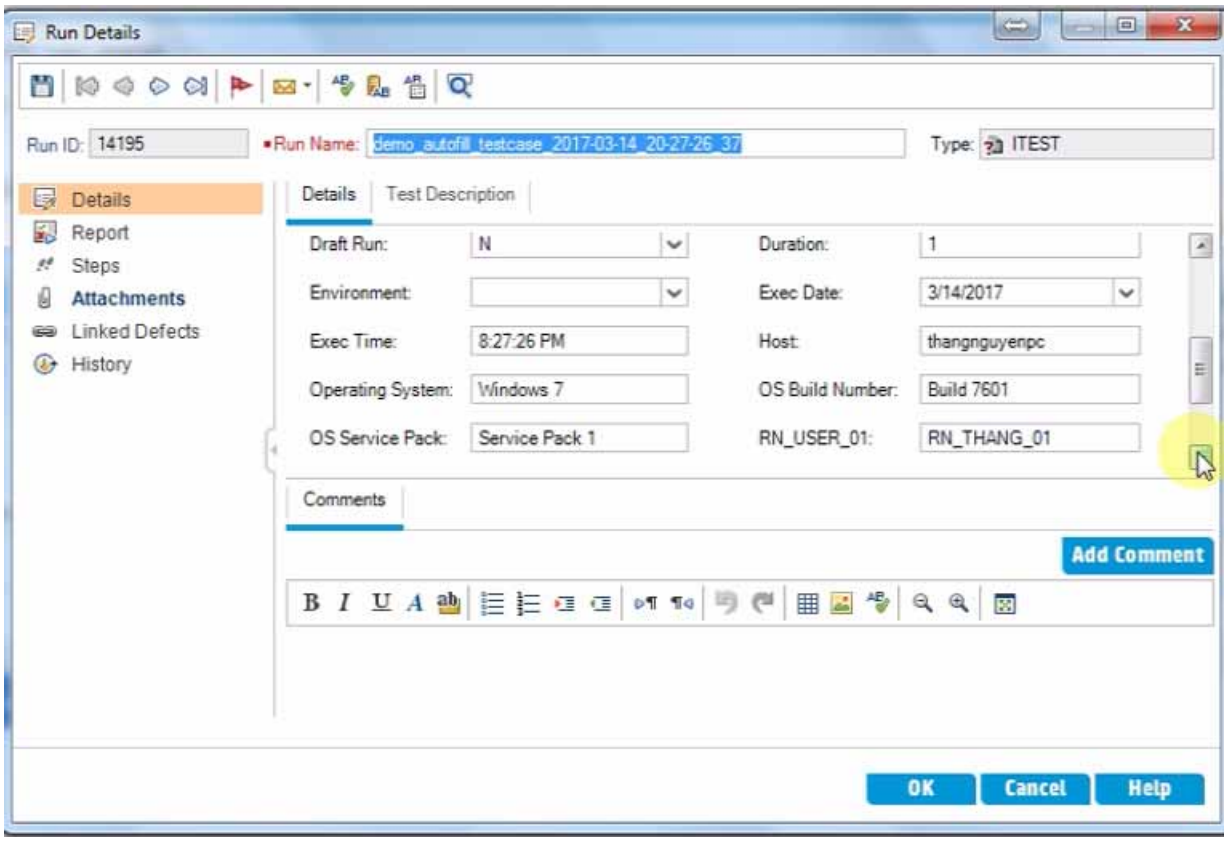
iTest RT: RT option > Param file > Test case param

Agent: Param file > Test case param > agent option

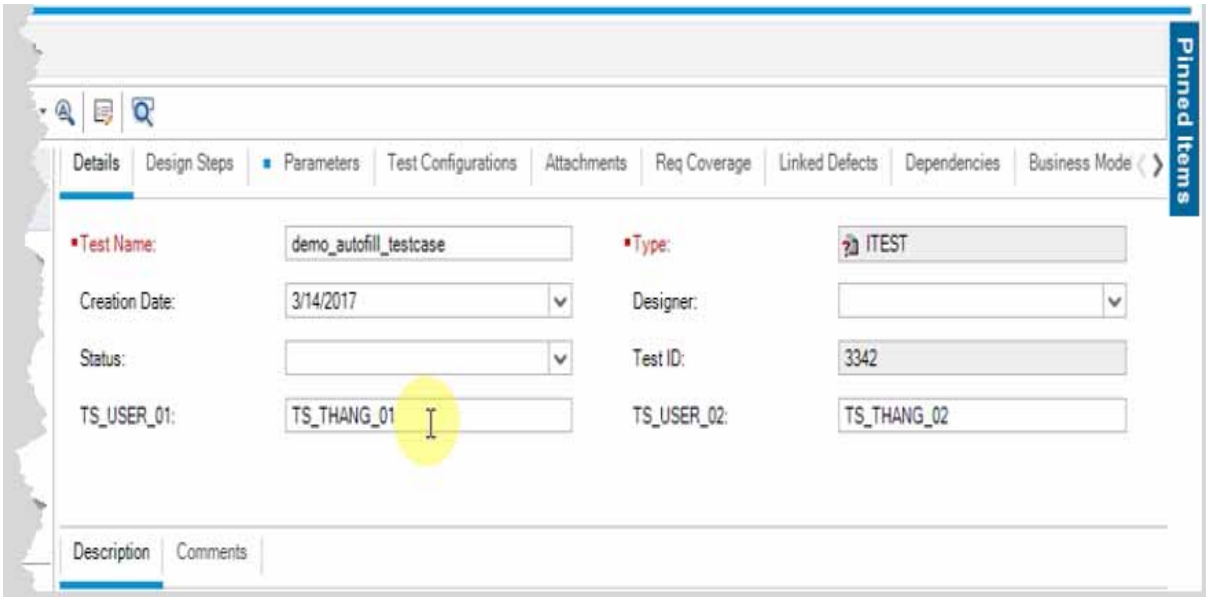
Test Case instance with the custom parameters mapped in HP ALM.



Test Case run with the custom parameters mapped in HP ALM.



Test Case details with the custom parameters mapped in HP ALM.



iTest Parameters for Quality Center

iTest allows HPQC options to be specified as test case parameters. The iTest Parameters for Quality Center is also supported by Velocity and iTestRT. In addition, iTestGUI, iTestRT, and Velocity will auto-publish test reports to the HP ALM server when execution completes and the required iTest parameters are provided.

Note These parameters are not supported in iTestCLI or when running test case from HP ALM server.

The parameter names of the HPQC arguments are not the same as the argument name supported via iTESTRT command line option. See "Quality Center options for iTestRT" on page 774).

qcServer	Specify the Quality Center server.
qcUser	Specify the username to connect to Quality Center.
qcPassword	Specify the password to connect to Quality Center.
qcDomain	Specify the Quality Center domain.
qcProject	Specify the Quality Center project.

qcReportFormat:	Uploads report format specified to the QC server.
	Both these URI formats are supported on iTestUI and RT:
	project:// and **file:/**
	Example:
	`project://my_project/<folder>/<filename>.<extension>`
	Use a single slash character after "`file:`" in the URI. For example:
	`file:/C:/Workspace/my_project/<folder>/<filename>.<extension>`

qcZip	Indicate True or False **True:** compresses the test report when publish to Quality Center server. **False:** Publishes the report format as follows depending on whether the report is published automatically or manually: • **Published automatically:** Uses the format indicated in **qcReportFormat** (above). • **Published manually:** Uses the default report format set in "ALM preference settings" on page 988. **Note** When publishing automatically, the format indicated in **qcReportFormat** takes precedence over the default parameter specified in "ALM preference settings" on page 988.
qcPublishChildren	Publish all nested (children) test reports to Quality Center.
qcPublish	Automatically publish the test report to Quality Center when test execution completes.
qcTestInstance	Specify the location of the existing test instance.
qcTestSet	Specify the test set that is contains the test instances.
qcTestPlan	Specify the location of the existing test plan.

Note Specifying the above test case parameters also allows Velocity to execute test cases imported from iTest and publish report to HP ALM without restarting the agent. In addition, Velocity allows changing value of the arguments between test case execution (for test cases imported from iTest).

Important: Follow these guidelines to use a parameter in both QC and iTest

- iTest can use both uppercase and lowercase letters in parameter names. In contrast, ALM parameter names must be all lowercase. To use a parameter in both QC and iTest, all letters in its name must be lowercase.
- iTest supports both hierarchical structures for parameters (where one parameter is a child of another parameter) and flat schemes (where all parameters are at the same level). In contrast, ALM supports only flat schemes. To use a parameter in both QC and iTest, do not create hierarchies of parameters in iTest.
- Because iTest supports parameters with identical names at the same level and QC does not, parameter names in iTest must be unique. Parameters with identical names at the same level in iTest are not published to QC.

Note In addition to the parameters listed above ("iTest Parameters for Quality Center" on page 966), see also Chapter 34, "Parameters" for detailed discussion of parameter used in iTest.

How parameters are added to Quality Center 10.0 and above tests

Parameters are attached to the test case. When you go into Test Lab to execute a testset (or a test instance), the QC **Parameter** dialog opens and, for any parameter, you can choose to use the current value or supply a new value.

To enable QC to execute iTest test cases, all QC tests of type **ITEST** include two predefined parameters (do not name other parameters with either of these reserved names.):

- **testbeduri** holds the URI of the iTest testbed to use (for parameter values) during execution
- **parameterfileuri** holds the URI of the iTest parameter file to use during execution.

Note If a parameter value is specified in QC, the QC value is used and not the value in the parameter file.

Important The Test Case parameters has higher priority and overrides settings on the Preferences page ("ALM preference settings" on page 988).

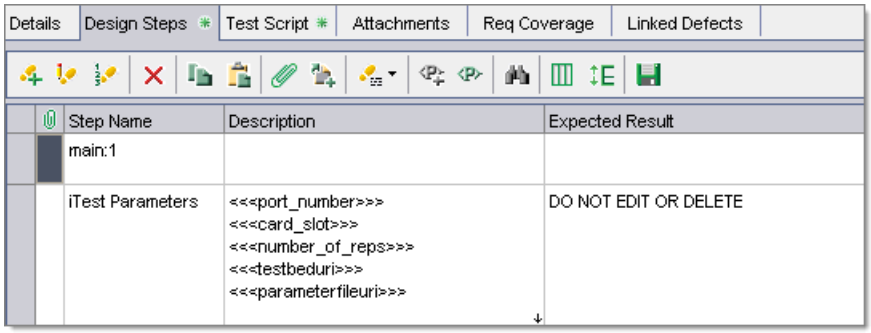
On the iTest **Quality Center** page, the **On publishing, replace Quality Center parameters with iTest parameters** check box operates as follows:

- Checked (default): When you publish the test case, the QC parameters are all deleted and replaced with the parameters from iTest. This ensures that the QC parameters are in sync with iTest test case parameters.
- Unchecked: When you publish the test case, parameters are added to QC if they do not already exist, otherwise the QC parameter **descriptions** are updated if appropriate.

How parameters are added to Quality Center 9.2 tests

Note QC 9.2 has technical limitations that do not allow iTest to add parameters to the default location in QC 9.2. Instead, the parameters are added to a special design step.

To ensure that it is clear to you which parameters are defined in iTest, QC displays all iTest-generated parameters together in a single design step. When you publish a iTest test case, QC adds a design step named **iTest Parameters as** the last step. The **Description** field contains the name for each parameter (each enclosed in angle brackets). The **Expected Result** for the step is set to **DO NOT EDIT OR DELETE** to remind you that this design step is used to store parameters.



Each time you publish a iTest test case to QC, the **iTest Parameters** step is overwritten with the current set of parameters as defined in iTest. (If the design step is ever deleted, a new one is created the next time the test case is published.)

To enable QC to execute iTest test cases, all QC tests of type **ITEST** include two predefined parameters: **TestbedUri** and **ParameterFileUri**, as shown in the example. The parameters hold the URI information that QC uses to configure iTest execution. Do not name other parameters with either of these reserved names.

Important The parameters in the **iTest Parameters** step are never exported from QC into the iTest test case. For information on setting parameter values when executing iTest test cases, see "Setting parameter values for test runs" on page 969.

Cases where parameters are added from QC tests to iTest test cases

Parameters are imported from a QC test into the associated iTest test case only when a new iTest test case is being created. There are two situations:

- You create a new iTest test case, associate it with an existing QC test that includes parameters
- While working on a test in QC (and there is not yet an associated iTest test case), you click **Edit Using iTest** and iTest creates a new test case.

Parameters that are defined in QC but not in iTest

For an existing iTest test case, you can define parameters in the associated QC test that do not exist in the iTest test case. The parameters are never added to the iTest test case, but the iTest test case can refer to such a parameter using the iTest **param** command.

Setting parameter values for test runs

When you execute a test from QC, you have the following options regarding parameter values:

- **Use QC parameters instead of iTest parameters**

  1. On the **Quality Center** page in the iTest Test Case editor, uncheck the **On publishing, replace Quality Center parameters with iTest parameters** check box.
  2. Publish the test case.
  3. In ALM, specify parameter values in the normal way.

     During execution, the QC value is used (overriding the value that is defined in iTest).

- **Use iTest parameters instead of QC parameters**

  1. On the **Quality Center** page in the iTest Test Case editor, check the **On publishing, replace Quality Center parameters with iTest parameters** check box. When you publish the test case, the QC parameters are all deleted and replaced with the parameters from iTest.
  2. Publish the test case.
  3. In ALM, leave the parameter values blank.

     When you execute, iTest gets each parameter value in the normal way; the value as defined in the parameter file and test case and in the associated testbed after merging/inheritance.

     Note If a parameter value is specified in QC, the QC value is used and not the value in the parameter file.

How itestcli uses parameter values set in QC

When you execute a test from QC, each parameter for which you have specified a value (as opposed to leaving the value blank) generates a command-line **-p** option to set the parameter value for execution.

Overview: Publishing iTest test cases to ALM server

The **Publish Test Case to Quality Center** wizard enables you to configure ALM server information and to update the appropriate design steps in a ALM test whenever you save a iTest test case (or when you click **Publish to Quality Center**). The wizard takes the following actions:

- The **Quality Center** server and domain are identified
- Your login information is specified (and typically remembered for as long as iTest is running — there are other options)
- The current iTest test case is associated with a test in ALM (under Test Plans). This information is used when you update design steps.
- Using the steps in the current iTest test case that are set as being identified with a design step in QC, the wizard then updates the design steps in the associated QC test. You can specify whether existing step names in QC should be overwritten.

- The test case is associated with the ALM test instance so that the appropriate test instance is updated with test report data when test case execution ends and you run the **Publish Test Report to Quality Center** wizard.

Once you have run the wizard, then you can execute tests using itestcli with the **-qc** command line option so that when test case execution finishes, itestcli publishes the test report data to ALM.

Storing test case information in ALM

To store project variables (project main, project path, and so on) in test execution issues:

1. Check **Store primary test case information in Quality Center** on the **Quality Center** page of the Test Case editor. (As described in "Configure settings that affect how test cases and test reports are published to QC" on page 958.)
2. To store the remaining parameters, set the appropriate parameters in QC based on the values in the test case configuration.
3. Publish the test case to QC

If you publish a test case to QC without checking the **Store primary test case information** check box, then, during execution, iTest executes a copy of the test case in a temp directory. Because execution occurs outside of the workspace, the project variables do not appear in test execution issues.

Publishing a single test case to Quality Center wizard

Note You cannot publish a iTest test case to a ALM test that is the **MANUAL** test type. Either:

- Convert the **MANUAL** test to **ITEST** using the test grid view

or

- Create a new test with the **ITEST** test type

The **Publish to Quality Center** wizard takes you through some simple steps that configure the connection between iTest and the ALM server, and link the definition of the test in ALM with the iTest test case and test reports. Follow this procedure:

1. If the test is open in ALM, then it is "locked" and cannot be overwritten. Close the test in ALM to "unlock" it.
2. In the Test Case editor, open the **Quality Center** page. Information about the ALM server appears on the page (the information is blank if you have not yet configured the connection to the server — more on that later). You will use this page to:
   - Configure the connection information for the ALM server
   - For new steps in the iTest test case: Update the ALM test information (that is, sync the steps in the iTest test case with the design steps in the ALM test)
   - If desired, open ALM in a separate browser so that you can perform any actions needed in ALM
3. Click **Publish to Quality Center**. The **Publish Test Case to Quality Center** wizard starts.

Login page

4 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next.**

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.
Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Domain and Project page

5 On the **Domain and Project** page, specify the following information for the test and then click **Next.**

The Domain and Project choices are obtained from the ALM server.

Domain	Specify the domain of the ALM server. If you do not specify a value, then iTest uses the **Default Quality Center domain** value that you specified on the **Preferences** page (Click **Window > Preferences**. On the **Preferences** page, click **Spirent > Quality Center**.) ("ALM preference settings" on page 988)
Project	Specify the project. We recommend that you use the name of the iTest project in which the test case is stored. If you do not specify a value, then iTest uses the name of the iTest project in which the test case is stored.

Test page

6 The **Test** page displays the tree from the ALM **Test Plan** page. Select the appropriate folder and then select the test.

Note If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Plan Folder	Click to add a test plan folder under the selected folder.
Add Test	Click to add a test under the selected folder.
Delete	In the ALM file structure, select a test plan folder or test and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test plan folder or test and then click **Edit** to modify its **Name** or **Description** text. For tests, you can also modify the **Status** and **Designer** settings. **Status**: Specify **Ready**, **Design**, **Import**, or **Repair**. Default: Ready **Designer**: Select the appropriate designer.
refresh	If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Test Instance page

7 The **Test Instance** page displays the ALM **Test Lab** tree view.

To cause itestcli to automatically publish test report data, you must associate the test case with a ALM test instance.

Select the appropriate folder, select the appropriate test set, and then select the test instance to update. Remember that you configured whether to include execution issues for nested steps for the iTest steps that are associated with ALM design steps (on the Test Case editor's **Quality Center** properties page).

To change the test instance that the iTest test case is associated with, check **Associate the iTest test case with the selected Quality Center test instance** and then select the test instance in the tree view.

Note If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Instance page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Add Test Instance	Click to add a test instance to the selected test set.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click Delete to delete it.

Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

8 Click **Next**. Alternatively, click **Finish** to publish the test case information without viewing the **Design Steps** page (described in the next step).

Design Steps page

9 The **Design Steps** page summarizes the information that will be written to the ALM test.

You have the option to modify the **Status** setting. Specify **Ready**, **Design**, **Import**, or **Repair**.

Review the settings and then click **Finish**. To make changes, click **Back** to move to the appropriate page.

Publishing multiple test cases to ALM (Batch publishing)

Note You cannot publish a iTest test case to a ALM test that is the **MANUAL** test type. Either:

- Convert the **MANUAL** test to **ITEST** using the test grid view

or

- Create a new test with the **ITEST** test type

Use the **Test Case Publishing** wizard to publish multiple test cases in a batch to the ALM server. You start out in the iTest Explorer by right-clicking a folder that includes multiple test cases and then launching the wizard. When you finish running the wizard:

- iTest creates a new test in ALM for each selected iTest test case.
- If the associated ALM test already exists, then the design steps are cleared and overwritten.
- If you selected the wizard's **Associate the iTest test case with the selected Quality Center test set** option (default), then when each test is created or overwritten in ALM, a test instance for the test is created in the appropriate test set.
- The ALM **Test Plan Location** and **Test Instance Location** are updated and saved in the iTest test case. This ensures that iTest test reports are published to the correct location in ALM.

Follow these steps to publish a batch of test cases:

1 If a test is open in ALM, then it is "locked" and cannot be overwritten. Close the test in ALM to "unlock" it.

2 On the iTest Explorer, select a folder that holds all the test cases to publish. Right-click and select **Publish Test Cases**. The **Test Case Publishing** wizard starts with an overview page that you can read and skip in the future.

Login page

3 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center**

page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next.**

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.
Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Domain and Project page

4 On the **Domain and Project** page, specify the following information for the test and then click **Next.**

The Domain and Project choices are obtained from the ALM server.

Domain	Specify the domain of the ALM server. If you do not specify a value, then iTest uses the **Default Quality Center domain** value that you specified on the **Preferences** page (Click **Window > Preferences**. **> Spirent > Quality Center** - "ALM preference settings" on page 988).
Project	Specify the project. We recommend that you use the name of the iTest project in which the test case is stored. If you do not specify a value, then iTest uses the name of the iTest project in which the test case is stored.

Test page

5 The **Test** page displays the tree from the ALM **Test Plan** page. Select the appropriate folder to publish the tests to.

Note If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Plan Folder	Click to add a test plan folder under the selected folder.
Delete	In the ALM file structure, select a test plan folder or test and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test plan folder or test and then click **Edit** to modify its **Name** or **Description** text. For tests, you can also modify the **Status** and **Designer** settings. **Status**: Specify **Ready**, **Design**, **Import**, or **Repair**. Default: Ready **Designer**: Select the appropriate designer.
refresh	If you use ALM to add a new folder or test while the **Test** page is open, click **refresh** to refresh the tree view.

Test Cases page

6 The **Test Cases** page lists all of the test cases in the folder you selected, including test cases in subfolders.

Select the test cases to publish to ALM and click **Next.** By default, all test cases are selected. Use **Select All** and **Unselect All** to speed up the selection process when selecting only some of the test cases.

Test Set page

7 The **Test Set** page displays the ALM **Test Lab** tree view.

Note To cause itestcli to automatically publish test report data, you must associate the test cases with a ALM test set.

---

8 Select the appropriate folder and then select the appropriate test set to update. Remember that you configured whether to include execution issues for nested steps for the iTest steps that are associated with ALM design steps (on the Test Case editor's **Quality Center** properties page).

To change the test set that the iTest test cases are associated with, check **Associate the iTest test case with the selected Quality Center test set** and then select the test set in the tree view.

Note If you use ALM to add a new folder or test set while the **Test Set** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Set page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click Delete to delete it.
Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Publishing test cases when switching workspaces or exiting iTest

When you exit iTest (or iTest switches to a different workspace at your request) the **Publish Test Cases to Quality Center** wizard reminds you when test cases that you have edited have not yet been published to ALM. (You can set a preference to auto-save unsaved test cases without prompting when running the wizard. See "ALM preference settings" on page 988.)

In addition, the wizard identifies ***incomplete*** test cases — test cases that have changed, but required additional information before publishing.

Between the two lists, therefore, the wizard identifies all test case that have previously been published to ALM and have been altered but not republished.

1. When the wizard starts, all test cases that are ready to publish are listed on the **Select Test Case** page. Select and unselect test case files as appropriate.
2. If the **Incomplete Test Cases** button is activated, then there are some test cases that cannot yet be published to ALM, either because of warnings or errors in the test case steps or because the ALM server or test are not fully specified on the **Quality Center** page on the Test Case editor. To view such files and the reason that each file is incomplete, click **Incomplete Test Cases**.

   To ensure that the incomplete test cases are published, cancel the wizard and edit the test cases as needed. Then restart the wizard.
3. Click **Publish** to publish the test cases to ALM and exit iTest or switch workspaces.
4. If the ALM server needs login information, then the **Publish** button is dimmed and you can click **Next** to open the **Login** page. On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you

had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next**.

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.
Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

5 Click **Publish**.

Overview: Publishing iTest test reports to ALM server

The **Publish Test Report to Quality Center** wizard enables you to upload two types of iTest data to ALM server: execution results for the test instance and iTest test reports files in the preferred format. iTest publishes the test report and its artifacts (the 'snapshots' for browser-based sessions like Web and Swing) as a zip file. Using a zip file maintains the folder structure so that images in the report appear as required.

Note The report format depends on whether the report is published automatically or manually:

- **Published automatically:** Uses the format indicated in the parameter **qcReportFormat** (see "iTest Parameters for Quality Center" on page 966)
- **Published manually:** Uses the default report format set in "ALM preference settings" on page 988.

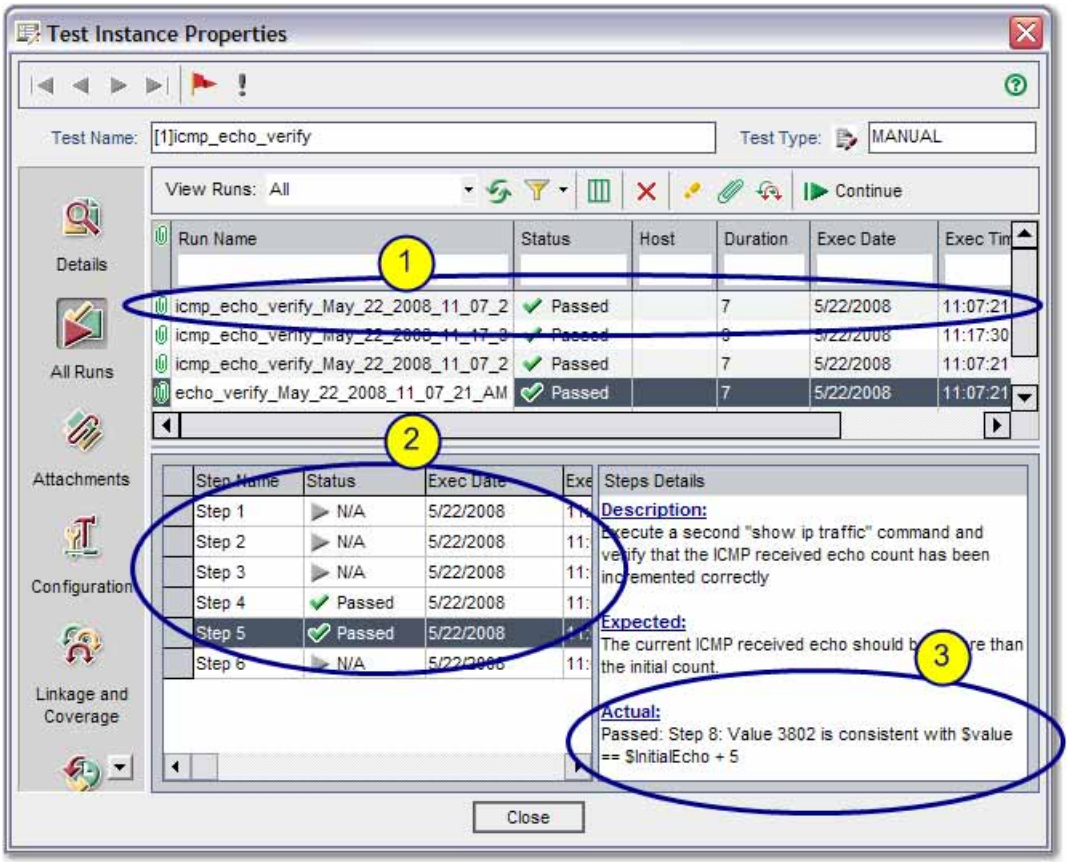
In addition to using the wizard to publish the test report data, itestcli can automatically publish the data when test case execution finishes.

Execution results for the test instance

The wizard updates the test instance execution results. As shown in the example, the text appears in the **Step Details** section of the **Test Instance Properties** page for the selected design step.

iTest associates execution issue severity with ALM design steps as follows:

iTest execution issue severity	Resulting "Actual Result" value in ALM
Information	**N/A**
Warning	—
Error (Fail) and Abort	**Failed**
OK (Pass)	**Passed**
For any step that does not have an analysis rule and you have set the iTest property for **Set Quality Center Status to Pass for steps with no analysis rules**	**Passed** See "Quality Center properties" on page 960 for details.
[not used]	**Not Completed**
For any step that does not have an analysis rule and you have ***not*** set the iTest property for **Set Quality Center Status to Pass for steps with no analysis rules**	**No Run** See "Quality Center properties" on page 960 for details.



1 The test run entry is named after the uploaded test report.

Status, **Duration**, **Exec Date**, **Exec Time**, and the global **Status** of the test run and test instance are updated from the test report, regardless of whether you associated iTest execution messages with steps.

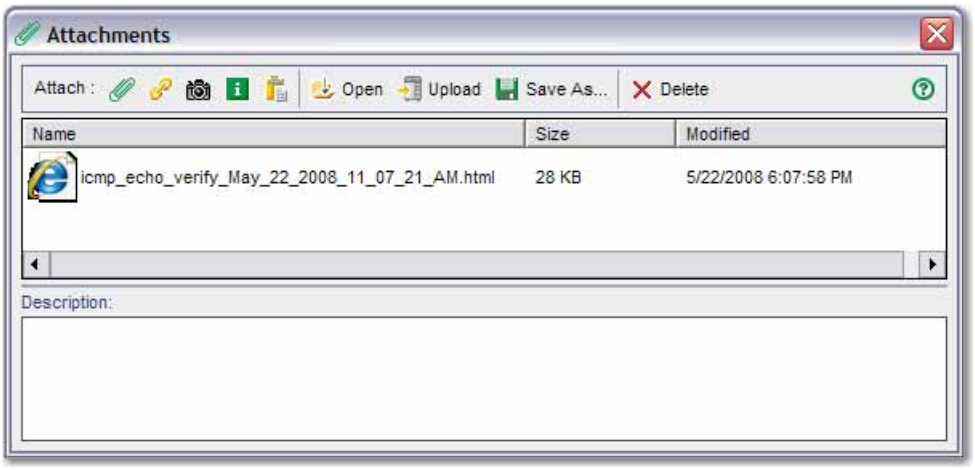
The **Tester** value is taken from login information.

2 If you associated iTest execution messages with steps, then **Status**, **Exec Date**, and **Exec Time** for each associated step are updated.

3 For each step that you associated with a design step, the text of the execution message appears here, for the **Actual** result.

iTest test reports as HTML-format files

iTest adds the reports as attachments in the ALM repository, associated with the test instance run entry.



Double-click a report to open it. Here's an example test report in HTML format:

Test Report

File generated at: Wed Feb 08 13:08:47 ICT 2017

Test case:	project://7501/test_cases/new_testcase2.fftc
Testbed:	
Parameters file:	project://my_project_2/session_profiles/param.ffpt
Owner:	
Test case ID:	
Test case name:	
Test case namespace:	
Execution started:	Wed Feb 08 11:58:51 ICT 2017
Execution completed:	Wed Feb 08 11:58:53 ICT 2017
Execution duration:	00:00:02.1
Report ID:	55117
Host:	PC1365
Group:	
Subgroup:	
Total report items:	6
Total issues:	6
- Pass/OK:	0
- Fail/Error:	4
- Warning:	0
- Information:	2
Result:	Fail

Parameters

Session	Parameter Name	Description	Value	Source
	qcPublish	mandatory	true	param.ffpt
	qcServer	mandatory	http://192.168.51.153:8080/qcbin/	param.ffpt
	qcDomain	mandatory		param.ffpt
	qcProject	mandatory		param.ffpt
	qcUser	mandatory	admin	param.ffpt
	qcPassword	mandatory	admin	param.ffpt
	qcZip	optional		param.ffpt
	qcReportFormat		project://resources/reports24/test_report_templates/PDF	param.ffpt
	qcPublishChildren		true	param.ffpt
	qcSet/TEST/USER-01		111	param.ffpt
	qcSet/TEST/USER-02		333	param.ffpt
	qcSet/Instance/USER-01		123	param.ffpt
	qcSet/Instance/USER-02		456	param.ffpt
	qcSet/RUN/USER-01		555	param.ffpt

Execution Issues

Index	Severity	Originator	Message	Location
	info	execution	Execution started	
1	fail	com.fnfr.svt.applications.spirent.testcenter.gui	Unable to open session : Failed to load Tcl package SpirentTestCenter	/procedures/0/steps/0
1	fail	com.fnfr.svt.applications.spirent.testcenter.gui	Internal interpreter must not be used for this application	/procedures/0/steps/0
2	fail	step executor	Session "s1" is not opened and therefore action "eval" is not allowed	/procedures/0/steps/1
2	fail	execution	Test case new_testcase2 has failed.	/procedures/0/steps/1
	info	execution	Execution completed (1s)	

Steps

Index	Action	Session	Start Time
	call		00:00:00.1
Command: main			

Publishing a single test report to ALM

The **Publish Test Report to Quality Center** wizard enables you to upload two types of iTest data to ALM server: iTest test reports in the preferred format and execution results for the test instance. iTest publishes the test report and its artifacts (the ‘snapshots’ for browser-based sessions like Web and Swing) as a zip file. Using a zip file maintains the folder structure so that images in the report appear properly.

Note The report format depends on whether the report is published automatically or manually:

- **Published Automatically:** Uses the format indicated in parameter **qcReportFormat** (see "iTest Parameters for Quality Center" on page 966)
- **Published Manually:** Uses the default report format set in "ALM preference settings" on page 988.

In addition to using the wizard to publish the test report data, you can execute tests using itestcli and automatically publish the data when test case execution finishes.

Tip Before executing a test case, use the **Quality Center** page on the Test Case editor or the batch publish option to publish the test case to ALM. This action sets the **Domain**, **Project**, and **Test Instance** location for the test case so that the wizard is pre-populated with the appropriate information.

Follow these steps to upload iTest test report data to ALM server:

1 Use either of the following methods to start the **Publish Test Report to Quality Center** wizard:

- In iTest, open the test report in the Test Report editor (the editor opens automatically when a test case finishes executing or when you double-click a test report in the Test Reports view). Click **iTest > Test Reports > Publish Test Report to Quality Center**.

  or

- In iTest, open the Test Reports view. Right-click a report and select **Quality Center > Publish Test Reports to Quality Center**.

Login page

2 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next**.

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.

Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Domain and Project page

3 On the **Domain and Project** page, specify the following information for the test and then click **Next**.

Domain	Specify the domain of the ALM server. **Note** To pre-populate the setting, you can specify the default **domain.** (Click **Window > Preferences**. > **Spirent > Quality Center** - "ALM preference settings" on page 988.)
Project	Specify the project.

Test Instance page

4 The **Test Instance** page displays the tree from the ALM **Test Lab** page. Select the appropriate folder, select the appropriate test set, and then select the test instance to update. Remember that you configured whether to include execution issues for nested steps for the iTest steps that are associated with ALM design steps (on the Test Case editor's **Quality Center** properties page).

Note If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Instance page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Add Test Instance	Click to add a test instance to the selected test set.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

5 Click **Next**. Alternatively, click **Finish** to upload the test results and HTML report without updating the design steps.

Finish page

6 The **Finish** page summarizes the information that will be written to the test instance.

Review the settings and then click **Finish**. To make changes, click **Back** to move to the appropriate page.

Publishing multiple test reports to ALM (Batch publishing)

Use the **Publish Test Reports to Quality Center** wizard to publish multiple test reports in a batch to the ALM server. You start out in the Test Reports view by right-clicking a folder that includes multiple test reports and then launching the wizard. The wizard publishes two types of data to ALM server: iTest test reports files in the preferred format and execution results for the test instance. iTest publishes the test report and its artifacts (the 'snapshots' for browser-based sessions like Web and Swing) as a zip file. Using a zip file maintains the folder structure so that images in the report appear properly.

Note The report format depends on whether the report is published automatically or manually:

- **Published Automatically:** Uses the format indicated in parameter **qcReportFormat** (see "iTest Parameters for Quality Center" on page 966)
- **Published Manually:** Uses the default report format set in "ALM preference settings" on page 988.

When you finish running the wizard:

- If you selected the default **Use Test Instance Location present in the test report** option, then each test report is published to the appropriate instance without any special action on your part.

  If any of the test reports did not include a value for the **Test Instance Location**, then the name of each such test appears in a text log. You can then use the Publish Test Case wizard to update the associated test case with the information so the report will be correctly published next time.

- If, instead, you selected the **Publish the reports to the selected Quality Center test set** option, then, if needed, a new test instance is created in the test set and the report is published to that location.

Tip The Test Reports view offers an option to display only the last report for every test case. You might use this option while iteratively developing and executing test case. When you finish, you could pick the last execution of each new test case that you developed that day and report the results.

Follow these steps:

1 Before executing a test case:

   a Use the **Quality Center** page on the Test Case editor or the batch publish option to publish the test case to ALM. This action sets the **Domain**, **Project**, and **Test Instance** location for the test case. This enables you to take advantage of the wizard's ability to

publish multiple test reports to the appropriate test instance.without you having to specify the test instance for each report.

b Use the Parameters page on the Test Case editor and set the **qcZip** parameter as true, which publishes the parent test reports and any child reports as a compressed file (see "iTest Parameters for Quality Center" on page 966).

Note Specifying the **qcZip** parameter as true for a parent test case and omitting to set the **qcZip** parameter as true for the child test cases, publishes a compressed report file with parent and child test reports (after completion of executing). However, specifying the **qcZip** parameter as true for child test case published only the child test case as a compressed file (after completion of executing).

2 In the Test Reports view, right-click multiple selected test reports and select **Publish Test Reports to Quality Center**. The **Publish Test Report to Quality Center** wizard starts. (To open the Test Reports view, click the arrow on the **Show View** button, select **iTest** and then select **Test Reports** in the drop-down list.)

Login page

3 On the **Login** page, you identify the Quality Center server and specify your login credentials (the credentials that you use when starting Quality Center). You have the option to have the wizard save your credentials for next time. If you had previously used the **Quality Center** page on the Test Case editor to publish the test case to Quality Center, then some of the information on this page should be populated.

Specify the values and then click **Next**.

Server name or URI	Type the hostname or IP address of the Quality Center server host. The **Default Quality Center server** value that you specified on the **Preferences** page appears in the list. If you specify a different server, then the preference value is updated. (To specify a default Quality Center server in iTest, click **Window > Preferences**. On the **Preferences** page, click **iTest > Quality Center**.) See "ALM preference settings" on page 988
User name	Type your user name.
Password	Type your password (if required). **Note** To run an iTest test case using the Quality Center **Automatic Runner**, a password must be defined for the account.
Remember my login information	Check the box to keep the values on this page for the next time you use the wizard. If you check the box, then choose one of the following options: **Until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always**: Save the values on disk. In this case, you can use the wizard at any time without having to re-type the server and login information.

Domain and Project page

4 On the **Domain and Project** page, specify the following information for the test and then click **Next**.

Domain	Specify the domain of the ALM server. **Note** To pre-populate the setting, you can specify the default **domain.** (Click **Window > Preferences > Spirent > Quality Center** - "ALM preference settings" on page 988.)
Project	Specify the project.

Test Set page

5 The **Test Set** page displays the tree from the ALM **Test Lab** page.

To cause itestcli to automatically publish test report data, you must associate the test reports with a ALM test set. By default, iTest attempts to use the **Instance Location** value in each test report determine the correct test set in order to populate the appropriate test instance. Select one of the following options:

Use the test instance location present in the test reports	Default. For each report in the folder, iTest uses the **Instance Location** value in each test report to populate the appropriate test instance.
Publish the reports to the selected Quality Center test set	Select this option to change the test set that the iTest test cases are associated with. If you select this option, then select the appropriate folder and select or add the appropriate test set in the tree.

Remember that you configured whether to include execution issues for nested steps for the iTest steps that are associated with ALM design steps (on the Test Case editor's **Quality Center** properties page).

Tip If you use ALM to add a new folder or test set while the **Test Set** page is open, click **refresh** to refresh the tree view.

Toolbar on the Test Set page

Actions that you take using the toolbar buttons take effect immediately on the ALM server.

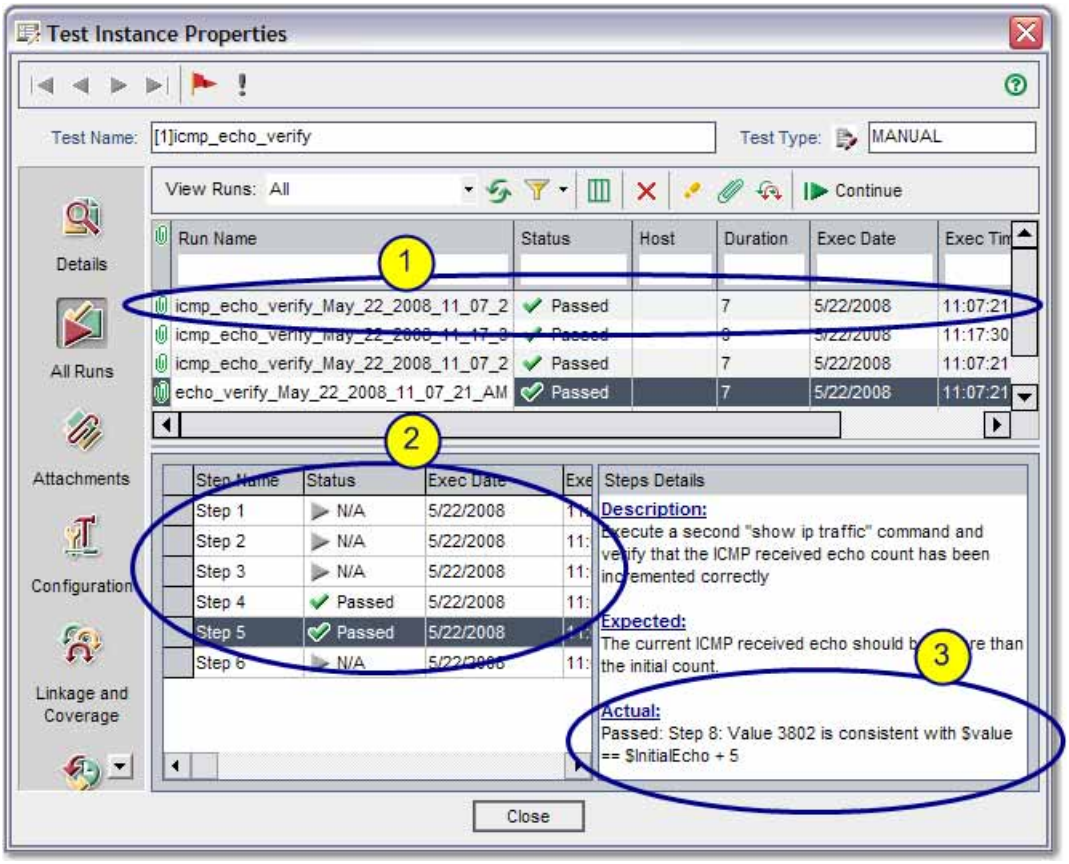
Add Test Lab Folder	Click to add a test lab folder under the selected folder.
Add Test Set	Click to add a test set under the selected folder.
Delete	In the ALM file structure, select a test lab folder, test set, or test instance and then click **Delete** to delete it.
Edit	In the ALM file structure, select a test lab folder or test set and then click **Edit** to modify its **Name** or **Description** text.
refresh	If you use ALM to add a new folder, test set, or test instance while the **Test Instance** page is open, click **refresh** to refresh the tree view.

6 Click **Finish** to upload the test results and HTML report.

After you click **Finish**, if any of the test reports cannot locate an appropriate **Test Instance**, then a warning dialog box lists the name of each such report.

Viewing the updated ALM report

When you finish working in the **Publish Test Report to Quality Center** wizard, you can view the results in ALM.



1 The test run entry is named after the uploaded test report.

Status, **Duration**, **Exec Date**, **Exec Time**, and the global **Status** of the test run and test instance are updated from the test report, regardless of whether you associated iTest execution messages with steps.

The **Tester** value is taken from login information.

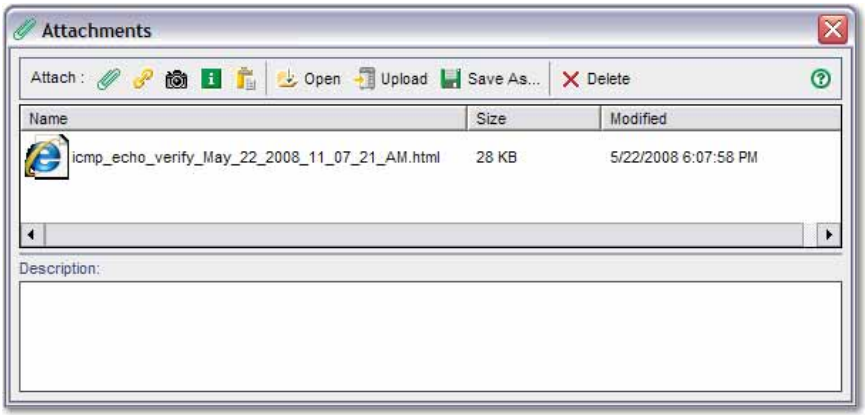
2 If you associated iTest execution messages with steps, then **Status**, **Exec Date**, and **Exec Time** for each associated step are updated.

3 For each step that you associated with a design step, the text of the execution message appears here, for the **Actual** result.

The iTest test report is saved in the selected format and is added as an attachment in the ALM repository, associated with the test instance run entry.

Click **Attachments** to view the list of the test report files.

Double-click a report to open it.



Viewing a iTest test report in ALM

To view a iTest test report, open the QC **Test Instance Properties** window.

Note The report is published to QC in the format specified in "ALM preference settings" on page 988 (default) or in the format selected "Setting parameter values for test runs" on page 969.

ALM preference settings

To view or edit preferences, click **Window > Preferences.** In the **Preferences** tree, click **Spirent > Quality Center.**

Default Quality Center server	Specify the ALM server to select by default on the Login page of the **Publish to Quality Center** wizards. Other servers may appear in the selection list.
Default Quality Center domain	Specify the ALM domain to select by default on the Login page of the **Publish to Quality Center** wizards. Other domains may appear in the selection list.
Login Information	Specify the default configuration for saving login credentials as it will appear on the **Login** page of the **Publish to Quality Center** wizards: **Remember my login information until I exit iTest**: Save the values only while the current instance of iTest is running. In this case, you can use the wizard multiple times without having to re-type the server and login information and the information is not stored on disk. The next time you use iTest, you must enter the information again. **Always remember my login information**: Save the values on disk (the password is encrypted). In this case, you can use the wizard at any time without having to re-type the server and login information. **Do not remember my login information**: Do not save credentials in any case. You will always have to type the information when using a **Publish to Quality Center** wizard. Default: **Remember my login information until I exit iTest** **Note** To run a iTest test case using the ALM **Automatic Runner**, a password must be defined for the account.
Run the Publish to Quality Center wizards in demo mode	Select to run the Test Case or Test Report wizards while not connecting to a ALM server. In demo mode, iTest populates all wizard pages with real-looking information to give you a realistic walk-through of wizard operation. In demo mode, no data is ever transferred to a ALM server. Default: unchecked
Show the introduction page for the Publish Test Case wizard	Select to start the wizard on a page that describes what the wizard does. Uncheck to skip the introduction page. Default: checked
Show the introduction page for the Publish Test Report wizard	Select to start the wizard on a page that describes what the wizard does. Uncheck to skip the introduction page. Default: checked
When exiting iTest or switching workspaces, save test cases and publish them to Quality Center	Select option so that when you exit iTest or switch workspaces the **Publish Test Cases to Quality Center** wizard auto-saves all edited and unsaved test cases (as described in "Actions that you take using the toolbar buttons take effect immediately on the ALM server. Publishing test cases when switching workspaces or exiting iTest" on page 976). Default: unchecked

Notify user if test requires custom fields	Select to notify users trying to publish a test case when there are custom fields in the QC test, that the fields will initially be set to default values, and that they must edit the values in QC. Default: checked
Publish the test report to Quality Center	Select to indicate that test reports should be published automatically to the Quality Center. When this option is selected, the publisher (click **Publish to Quality Center** action) automatically fetches any missing Quality Center Server information required for publishing reports to the Quality Center. iTest fetches the information you stored in the Quality Center as described in "Login page" on page 948.
Default Quality Center Project	Specify the ALM project to which the reposts should be published. If no project is specified, all reports are published to the default Quality center location.
Publish all children's test report to Quality Center	Select to indicate whether all children's test reports should be published to the project location
Report Format	Select the default report format to publish reports.Select the required format from the list: HTML, Text, XML, XML_Raw and PDF **Note** When publishing automatically, the format indicated in the test case (parameter **qcReportFormat**) takes precedence over the default parameter specified here.

CHAPTER 48

Ranorex Sessions

Overview

Ranorex Test session provides ways of automating the Windows, Web, and Mobile UI applications. The seamless testing of these iTest functionalities is achieved by iTest integration with Ranorex application (www.ranorex.com). In addition, Ranorex provides the ability to process images, which is useful for desktop or web applications that have hidden objects.

iTest also supports simultaneous execution of multiple Ranorex sessions each running on a different mobile device.

The maximum number of mobile devices used to run multiple Ranorex sessions depend on your resource capacity.

This chapter includes these topics:

Note Before you begin, see also "Ranorex action commands" on page 991 to understand the action commands in Ranorex.

General recommendations when running Ranorex test case

The following is recommended to avoid unexpected results when running Ranorex test case:

- Run only one instance of the application-under-test at a time (unless the test requires more than one instance of the application-under-test running simultaneously).
- Close all applications that are not required as part of the test.

Note In addition, it is recommended you *not* Extend your screen or multi-task (with your mouse or keyboard) when Ranorex test case is running:

Ranorex action commands

The following lists the action commands available when testing Ranorex sessions.

Important Some Ranorex action commands have changed. To migrate test cases from Ranorex 5.x to 8.x, edit the test case as follows in the text editor.

- Replace all **Key Sequence** with **Key sequence**.
- Replace all **Key Shortcut** with **Key shortcut**.
- Replace all **Wait For** with **Wait for**.
- Replace all **Run Application** with **Run application**.
- Replace all **Close Application** with **Close application**.
- Replace all **Mousewheel** with **Mouse wheel**.
- Replace all **Set Value** with **Set value**.

Save test case and reopen with iTest Test Case Editor.

Note For details, refer to https://www.ranorex.com/help/latest/ranorex-studio-fundamentals/actions/introduction/.

Click	Adds a new mouse action item at the current position. This action is typically used for button clicks. Use the mouse action for 'Up', 'Down', 'Click', 'Double-Click' and 'Move'-actions. **Note** This action requires a RanoreXPATH defined in a target.
Close application	Use action for closing applications and web sites. If the 'Close Method' is set to 'CloseWindow', the application is attempted to close. If the parameter 'Grace Period' is set to a value greater than 0 ms, the process will be killed after the 'Grace Period', if the application fails to close. If 'Close Method' is set to 'KillProcess' the application's process is killed immediately, even if the value of the value of 'Grace Period' is > 0ms. **Note** This action requires a RanoreXPATH defined in a target. Example: /form[@processname='notepad' and @instance='0']
Deploy Android app	The "Run Mobile App" action which starts the instrumented APK file on the selected device. When using this action you can specify a 'Device' and an 'App'. For more information see Android Testing - Record your Android Test.
Deploy iOS app	The "Run Mobile App" action which starts the instrumented APK file on the selected device. When using this action you can specify a 'Device' and an 'App'. For more information see Android Testing - Record your Android Test
DoubleClick	This mouse action can be used for 'Up', 'Down', 'Click', 'Double-Click' and 'Move'-actions. This action is typically used for button clicks so a repository item assignment is required.
Down	This mouse action can be used for 'Up', 'Down', 'Click', 'Double-Click' and 'Move'-actions. This action is typically used for button clicks so a repository item assignment is required.

Get Attribute	use the action to get values of various object attributes in the UI. You may also use the **Get Attribute** action to search for an image or compare a known screen image to the UI under test. For more information, see "Capturing and Validating Elements" on page 999. **Note** This action requires a RanoreXPATH defined in a target.
Key shortcut	Use action for executing key shortcut actions. In addition to the shortcut actions, you may also specify the key codes **Press**, **Up** and **Down**. **Note** This action requires a RanoreXPATH defined in a target.
Key Sequence	Use action to execute or to simulate a key sequence on a specified repository item. This action is typically recorded in a form filling scenario (e.g., username field in login process). **Note** This action requires a RanoreXPATH defined in a target.
Mobile Key Press	Adds a new mobile key press action (e.g. \{BACK\}, \{MENU\}).
Mouse wheel	Adds a new mouse wheel action item. Use the mousewheel action to specify as **Vertical** or **Horizontal** direction. You may also specify a wheel-delta, which is 120 by default.
Move	The action moves the mouse cursor to specified location. Location may be specified as fixed or proportional. Fixed can be absolute x;y pixel coordinates (integer values) or any one of the following named locations: Center, CenterLeft, CenterRight, UpperCenter, UpperLeft, UpperRight, LowerCenter, LowerLeft, LowerRight. The default location is Center. Proportional location is specified as x;y decimal values. Example: .25;.25 **Note** If no RanoreXPATH is defined in the target, the **Move** action is relative to the screen. When RanoreXPATH is defined, the **Move** action is relative to the object defined in the target.
Open Web Browser	Opens a browser and navigates to the given URL. Use the parameter browser for choosing a browser (e.g. IE, Chrome, Safari, or Firefox). **Note** Ensure that you enter the required URL in the description field on the Test case editor.
Run application	Runs an application at the given directory and file path. Use this action to run an application with file name and path specified in **Step properties > General > Command**. Manually enter the file name and path as follows. In **Step properties** section, click **General** and enter the file name and path in the **Command** field. **Note** Ensure that you enter the required path in the description field on the Test Case Editor. Example: C:\\Windows\\notepad.exe
Run Mobile App	Runs an application on a mobile device (e.g., iOS and Android devices).
Set value	Adds a new **set value** action to set an attribute value (e.g. 'Text' for a UI element of type text). Use action for setting the values of repository items. Depending on the assigned repository item, the available attributes might be different.
Swipe	The touch event action at the current position (**Touch Start**, **Touch Move**, and **Touch End)** simulates a drag gesture in Ranorex.
Touch	The touch event action at the current position (a **Long Touch**) typically opens a context menu.

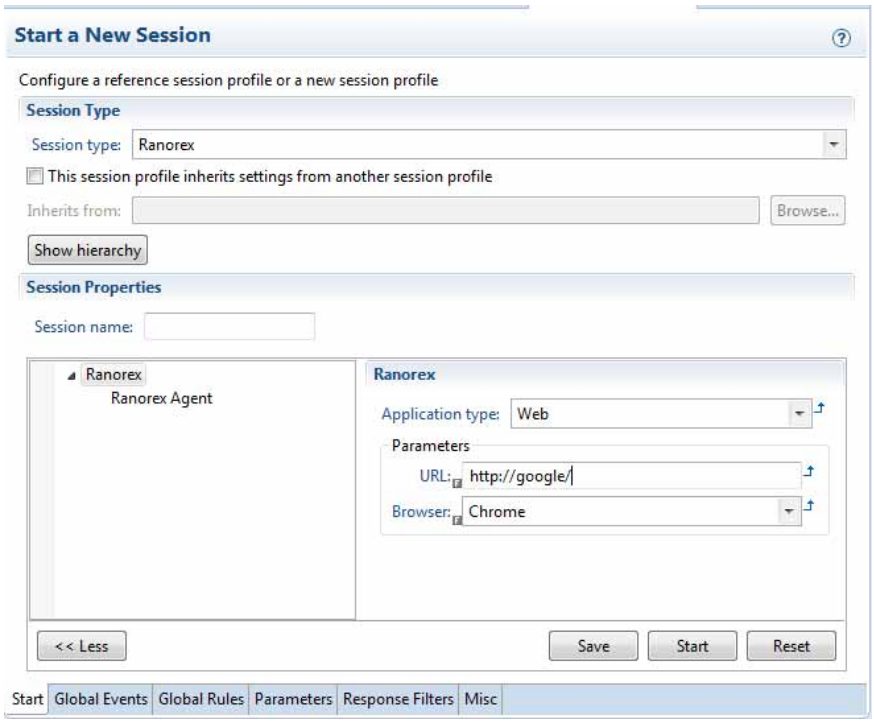
Up	Adds a new mouse action item at the current position. This action is typically used for button clicks. The action bring a window to the top and gets focus on the current window. Use the mouse action for '**Up**', '**Down**', '**Click**', '**Double-Click**' and '**Move**' actions. **Note** This action requires a RanoreXPATH defined in a target.
Wait for	You can use this action to **Wait for** the target item to appear within the specified timeout. **Note** This action requires a RanoreXPATH defined in a target. Example: /form[@processname='notepad' and @instance='0']

Creating Ranorex Session Profile in iTest

- Create a Ranorex session profile.
- Select the **Ranorex** session from the session type list. Click **More** and then **Ranorex**. Select **Application Type** from the list: **Desktop**, **Mobile**, **Web**. By default, the **Web Application Type** is selected. See these topics for creating Ranorex Sessions.
  - "Ranorex Web Session" on page 994,
  - "Ranorex Desktop Session" on page 995,
  - "Ranorex Mobile Session" on page 996.
- Click **Ranorex Agent** and enter the Ranorex Agent port to run multiple Ranorex sessions on different ports. See "Configure Ranorex Agent Port" on page 997.

Note The **Parameters** options displayed vary depending on your selection.

Ranorex Web Session



Note To improve recording process, use Ranorex instrumentation wizard (included in Ranorex installation) to install Ranorex browser plugins.

For a web-based application type, you must specify **Web**, **URL**, and **Browser**.

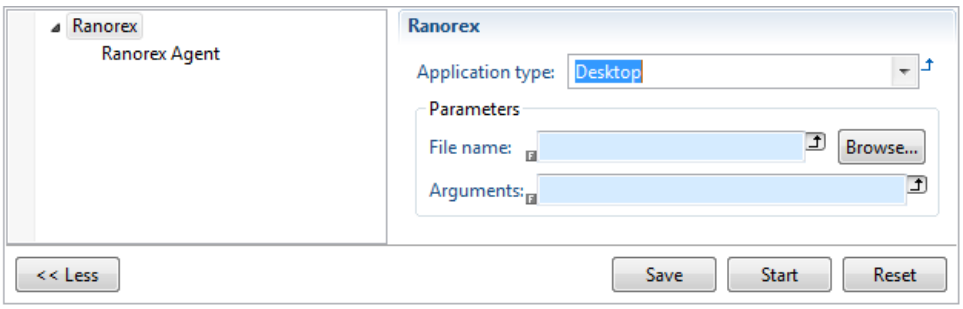
Application Type	Select **Web** from the list:.
URL	Specify the URL of web page at which to run the browser.
Browser	Choose the browser type from the drop down list. The Google Chrome, Mozilla Firefox and Microsoft Internet Explorer are supported. Save session and click **Start**. See "Using Ranorex Recorder" on page 997 for details • A new browser window opens with the specified URL in the session properties. • Ranorex recorder appears in the right bottom corner of the Window. • After that, all actions performed, will be captured in iTest. • Click **Stop** on the recorder when you are done and close the session in iTest. **Note** All major browsers are supported. See "Configure Ranorex Agent Port" on page 997 for configuring **Ranorex Agent** port to run multiple Ranorex sessions.

Ranorex Desktop Session

For a desktop application, select **Desktop** and provide **File name** and optional **Arguments**.

Application Type	Select **Desktop** from the list:.
	A new application instance opens using the specified path in the session properties. • Ranorex recorder appears in the right bottom corner of the Window. • After that, all actions performed, will be captured in iTest. • Click **Stop** on the recorder when you are done and close the session in iTest. **Note** See "Using Ranorex Recorder" on page 997 for details
File name	Path to an executable file to tested. **Note** File name cannot be blank. The Ranorex session will not start without a file name. **Important** Enter File name as **cmd.exe** and Arguments as **/c exit**, if you wish the application to be started by another process or If you do not wish for Ranorex to start the application.
Arguments	(Optional) It contains arguments to run with.

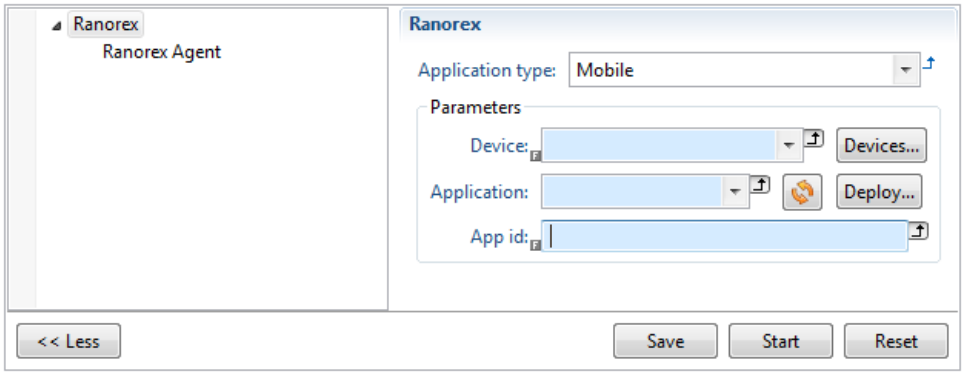
See "Configure Ranorex Agent Port" on page 997 for configuring **Ranorex Agent** port to run multiple Ranorex sessions.



Ranorex Mobile Session

For mobile application, select **Mobile** and provide details as listed below.

Application Type	Select **Mobile** from the list:.
Device **Application** **App id**	Select the mobile device, application and specify an App id. Use these buttons as follows. • **Devices**: Click to open the device manager. Allows managing mobile devices on the current environment. For details on how to setup mobile devices see these documents: iOS Testing and Android Testing. • **Update**: Click to retrieve application list from the selected device. • **Deploy:** Click to run Instrumentation wizard to deploy a new mobile application onto the device. **Note** See "Configure Ranorex Agent Port" on page 997 for configuring **Ranorex Agent** port to run multiple Ranorex sessions. iTest opens the specified application on the selected mobile device. See "Using Ranorex Recorder" on page 997 for details. **Note** It may take some time to connect to the device and start the recording. • Ranorex recorder appears in the right bottom corner of the Window. • After that, all actions performed, will be captured in iTest. • Click **Stop** on the recorder when you are done and close the session in iTest. **Note** Do not close the application or lock the device, as this will terminate recording process.



Note Ranorex 8.0 does not support recording of mobile action steps.

If the status of a mobile device displays as not available in the Ranorex "Manage Devices" dialog, follow these steps.

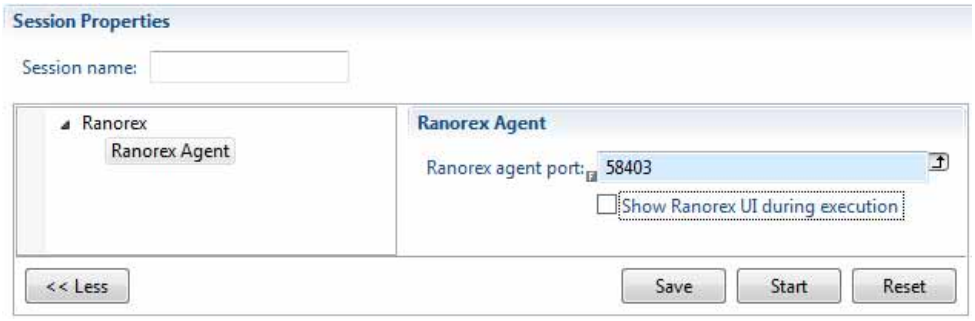
- Check Ranorex Services settings on the mobile device and ensure that the "Search system apps" option is disabled.
- Force shutdown Ranorex Services and open it again.

Note The "Search system apps" option may be enabled by default in your environment, which prevents from connecting to the device from ranorex recorder.

Configure Ranorex Agent Port

Configure Agent Port to indicate the port used to run Ranorex session.

Click **Ranorex Agent** on the **Start a New Session** window, in the **Session Properties** section.



Ranorex agent port	Indicates the port used to run a Ranorex session.
Show Ranorex UI during execution	Indicates whether the native Ranorex UI is visible during test execution. **Note** When executing multiple Ranorex Sessions, make sure that this option is **not** selected. That is, the native Ranorex UI is not shown.

Using Ranorex Recorder

To use the Ranorex session to capture, validate, analyze, and replay the Web/Flash, GUI, and Mobile applications. Follow these steps below.

Note Ranorex 8.0 does not support recording of mobile action steps.

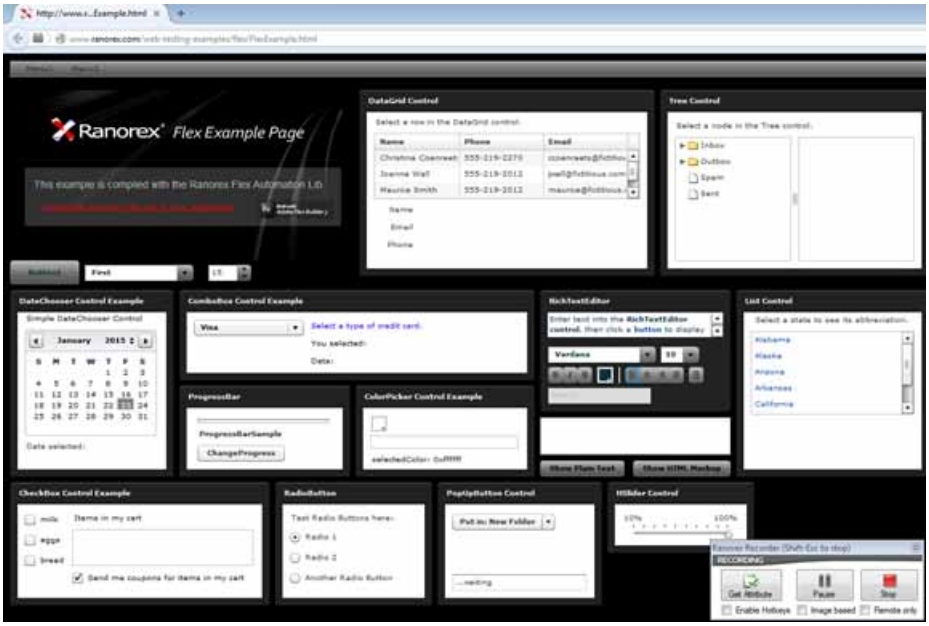
Step 1 Open a Ranorex session

Open a Ranorex session, for example, session created as described in "Creating Ranorex Session Profile in iTest" on page 994).

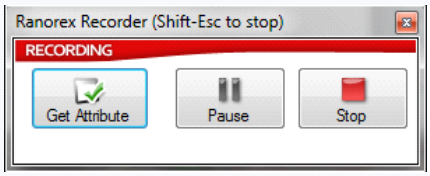
Step 2 Start Ranorex Session

Click the **Start** button to run Ranorex session.

Then perform the required test activities on the target Flash/Flex application page. All your actions will be captured by the Ranorex Recorder and sent to iTest, which will be recorded in the iTest Capture View and Capture Database.



When you start a Ranorex session, along with the Flash/Flex application in browser, the Ranorex Recorder Control Panel pops up with 3 buttons on the panel: **Get Attribute**, **Pause**, and **Stop**.



- Click **Pause**, to avoid recording any unwanted interaction with applications.

  The Ranorex Recorder Control Panel allows you to toggle between capture and pause by using the **Pause** button. Clicking **Pause**, changes the panel status (the **Continue** button becomes available) and the recording functionality is paused temporarily.
- Bring the application-under-test into the foreground.
- Click 

to resume recording.

  You may need to interact with your application until you reach the window that contains the object/image you wish to capture.

Note It is necessary to verify that you are on the correct page. Especially, when relying on mouse clicks at X-Y coordinates or a specific number of down/up keys, to ensure that you are clicking or keying to get to the right object.

- Click  to retrieve the element attributes.

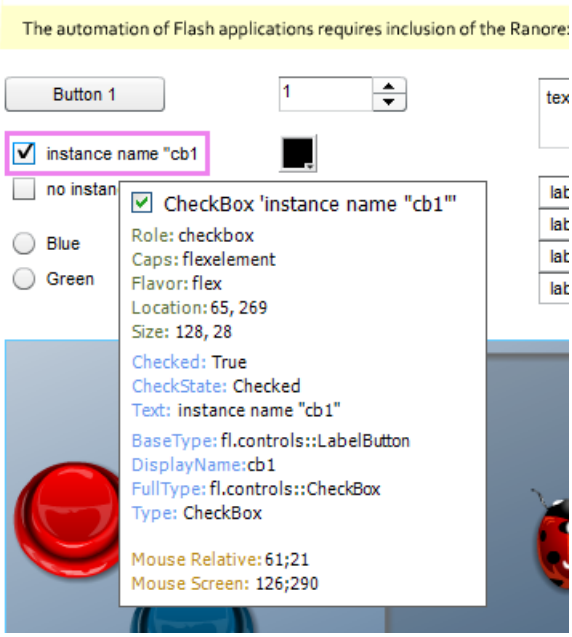
  The **Get Attribute** action is a special user action used to get attributes of the controls under test.

Step 3 Capturing and Validating Elements

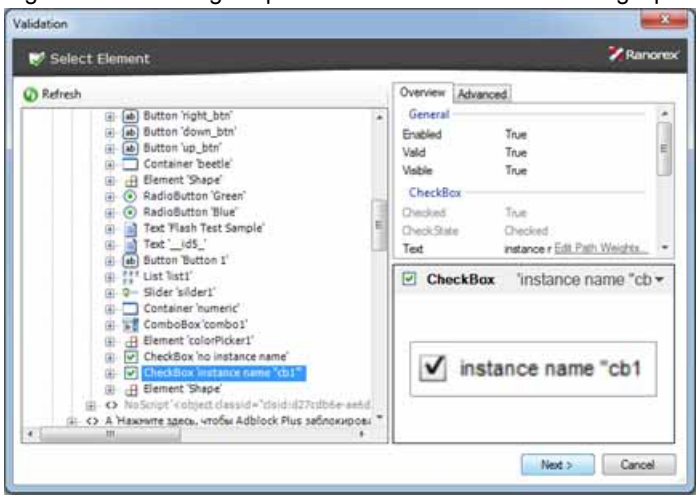
Use the **Get Attribute** command provided by Ranorex session to capture and compare an image under test. Follow these steps to capture object attributes.

Capture any object attributes in Web/Flash, Desktop applications:

- Click **Get Attribute** on the recorder and the recorder will enter into the **Validation Mode**. (See "Start Ranorex Session" on page 998.)
- Click to select any element (e.g., instance name 'cb1').

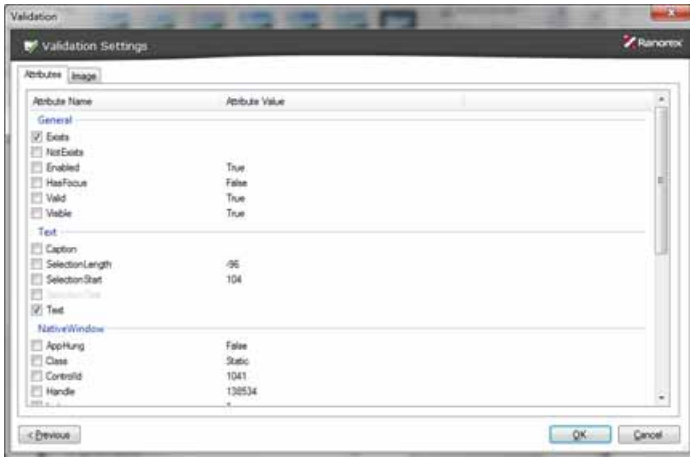


Click the area selected (e.g., instance name 'cb1'). A list of targets will be displayed. The element you select will be highlighted and the image captured will be shown in the lower right pane as illustrated below.



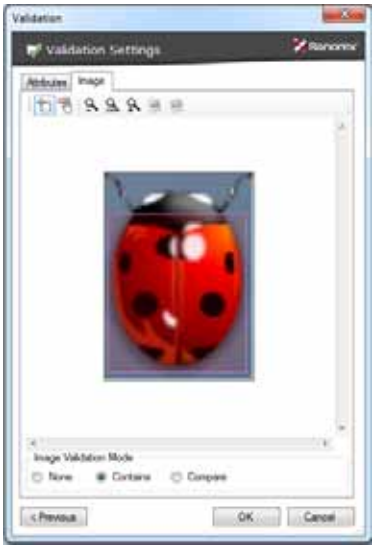
- Click **Next**. A list of attributes will be displayed. The Validation window with the selected element appears (for image-based elements captured, many of these values have no use). See "Capture image " below.

Note A new **Get Attribute** step will be recorded in iTest with the selected attributes.



Capture image

- Click **Get Attribute** on the recorder and the recorder will enter into the **Validation Mode**.
- Click to select any element and the Validation window appears with the selected element.
- Click **Next** and go to the **Image** tab (see illustration above).
- Select the desired image area and the image validation mode.
  - **None**: Indicates no validation is required.
  - **Contains**: Indicates that the image must contain only the pixels which you highlighted.
  - **Compare**: Indicates that every pixel within the image must match the image you read at runtime.
- Click **OK.**



Note A new **Get Attribute** step will be recorded in iTest with the selected attributes

Capture object attributes in Mobile applications

- Click **Get Attribute** on the recorder and the recorder will enter into the **Validation Mode**. (See "Start Ranorex Session" on page 998.)
- Select an element from the element tree on the left
- Click **Next** and select attributes to retrieve.
- Click **OK**

Name	Value	Type
structure		element
attributes		element
Checked	True	element
mapped		element

Note A new **Get Attribute** step will be recorded in iTest with the selected attributes

- The Attributes will also appear in the structured response of the **Get Attribute** step. (see illustration in the right column).

You can now easily add Analysis rules to the step and Validate your tests.

Step 4 Click Stop to end capture

When you complete testing, On the **Ranorex Recorder**, click  to end capture.

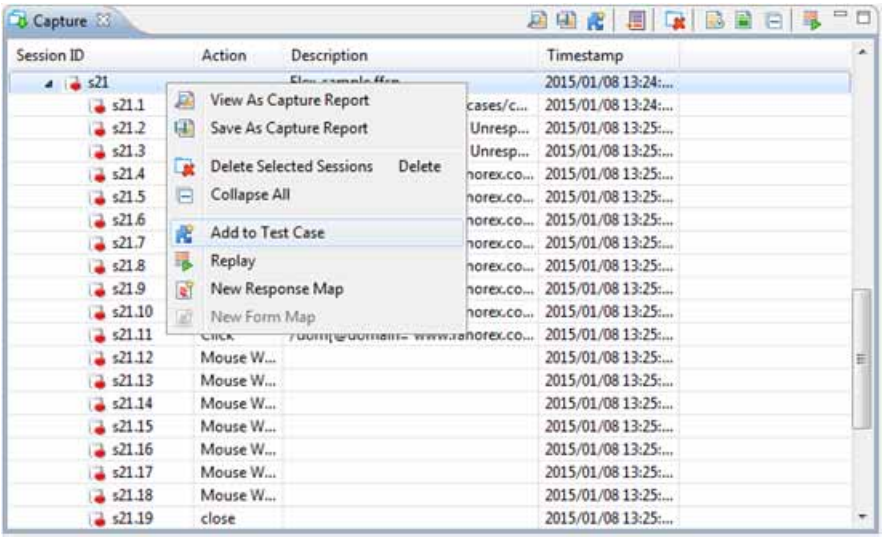
Go to "Generating iTest Test case" below.

Generating iTest Test case

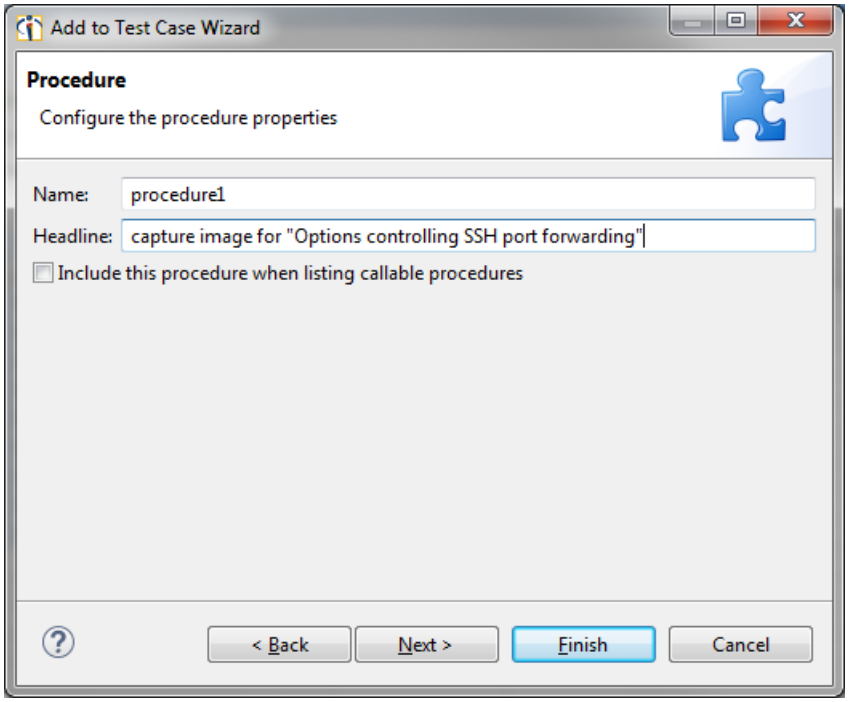
Perform these tasks in iTest Session window after you capture the required attributes.

In the iTest Session window, open/select Show View/Capture. This will display all your recorded operations on Flash/Flex application.

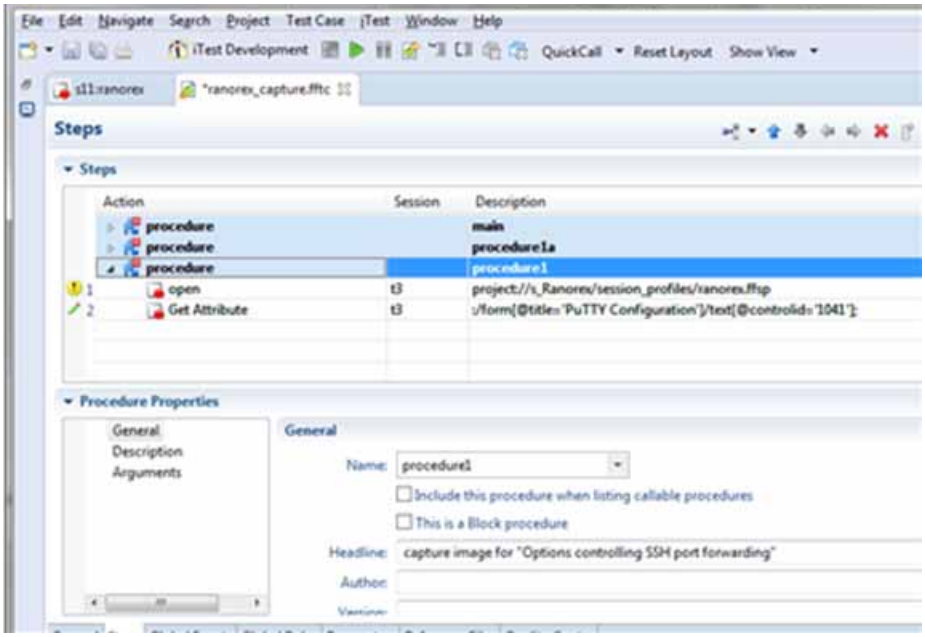
Highlight the captured session and click "Add to Test Case".



Since images have no readable properties such as a title or label, it is recommend to enter a Headline, so that you can easily identify the captured item.



In the iTest Session window, go to the procedure just captured and find the **Get Attribute** action.



Go to the **Get Attribute** step.

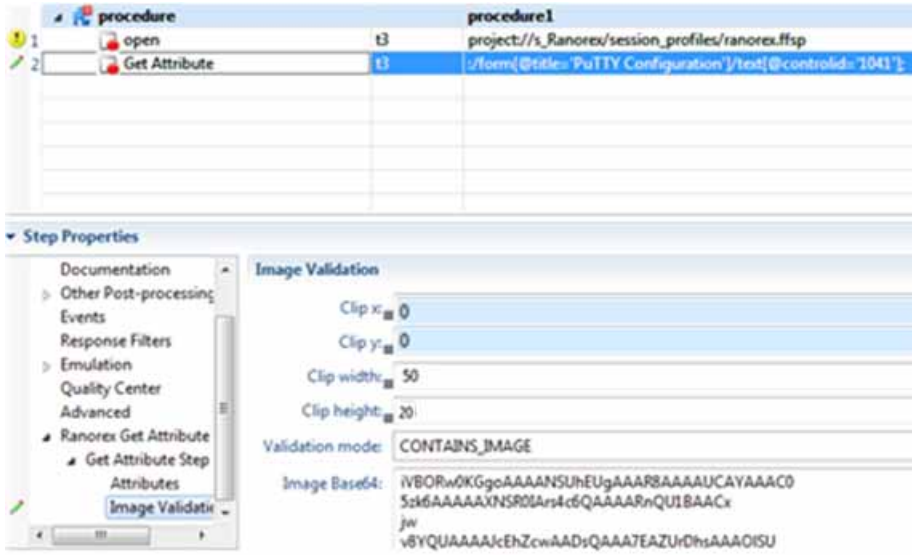


Image Properties:

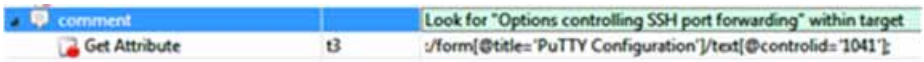
Clip x; **Clip y**; **Clip width**; **Clip height**: Correspond to the image you selected at capture time.

Validation mode: Indicates how to validate the image: (**None** indicates that no image is included for validation).

- **CONTAINS_IMAGE:** checks if target element contains part (specified by clip rectangle) of specified image
- **COMPARE_IMAGE**: check if target element image and specified image are the same

Image Base64: Encoded to Base 64 image data, which was saved during recording.

Note The Image Base64 is a text representation of the image. As there is no easy way of identifying the image, it is recommended that you add a step comment.

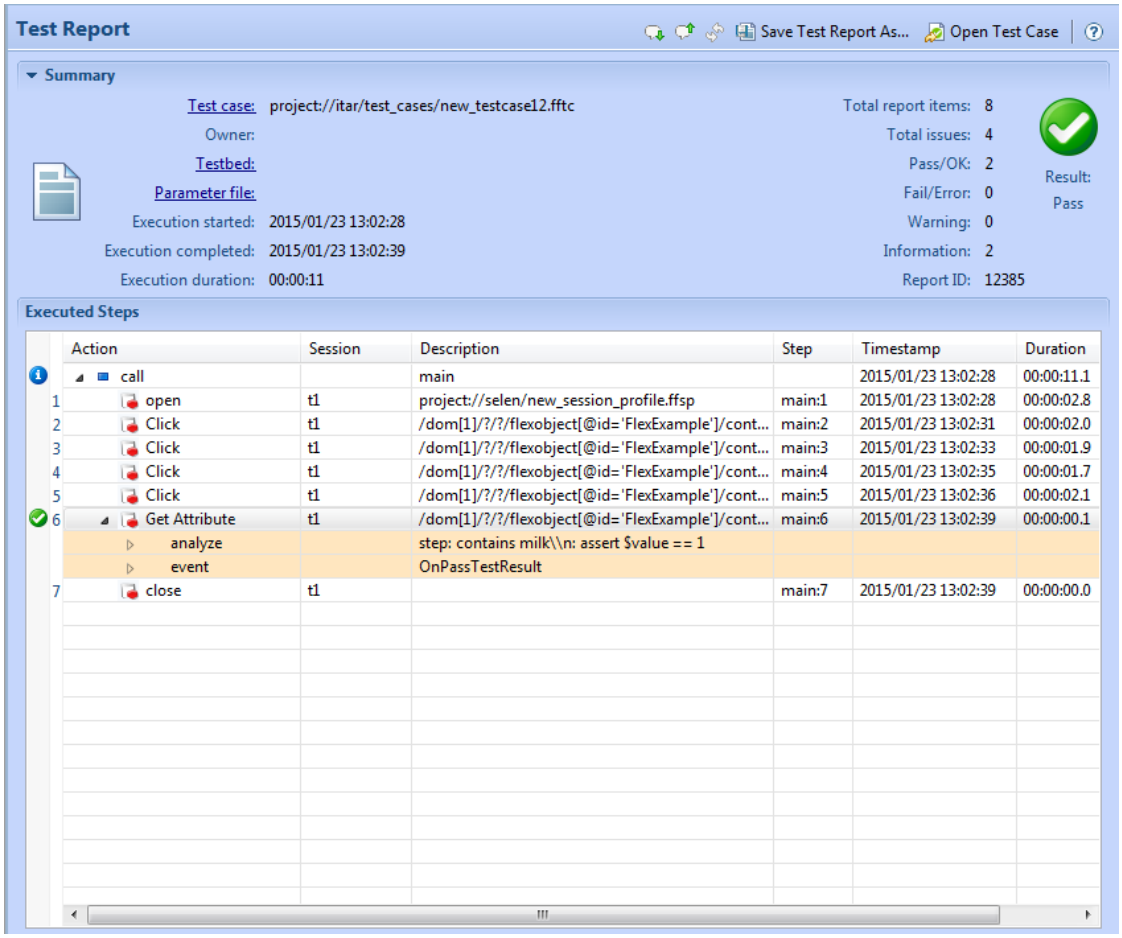


Validating in iTest

Based on the step response, you can add the analysis rule for any step in iTest. For example, after capturing the image and running the test case, you may add analysis rule for Get Attribute response.

Replaying Test Case

When the test case is ready, you can replay the test case as any the other sessions. When the playback is complete, the test report will be generated.



Running Multiple Ranorex Sessions

iTest supports simultaneous execution of multiple Ranorex sessions each running on a different mobile device. In addition to the illustrated example, iTest Ranorex session supports execution of the following configurations:

- Single test case executing against a single mobile device
- Single test case executing against two or more mobile devices
- Two or more test cases simultaneously executing against a single mobile device
- Two or more test cases executing simultaneously, with each test executing against two or more mobile devices

The maximum number of mobile devices that can be tested simultaneously depends on available resources..

Note iTest supports running either one Web or Desktop session, and one or more Mobile sessions at the same time.

Tasks to run multiple Ranorex sessions

Note The example below illustrates setting up a test case to run multiple Ranorex sessions.

Perform these tasks to run multiple Ranorex sessions.

1. Set up Ranorex Mobile sessions. See "Creating Ranorex Session Profile in iTest" on page 994. (A minimum of two different sessions for this example).

   Make sure to set each Ranorex session to use different Ranorex Agent ports. See "Configure Ranorex Agent Port" on page 997.

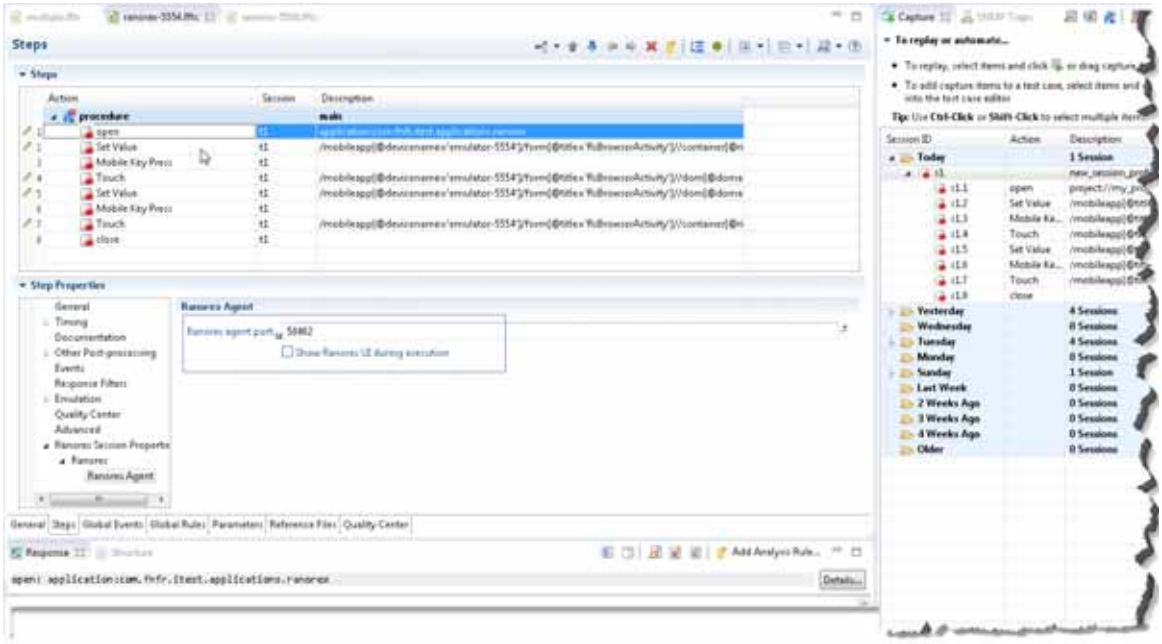
2. Capture and validated elements in Ranorex UI. See "Using Ranorex Recorder" on page 997.

3. Generate test case using the steps described in "Generating iTest Test case" on page 1001.

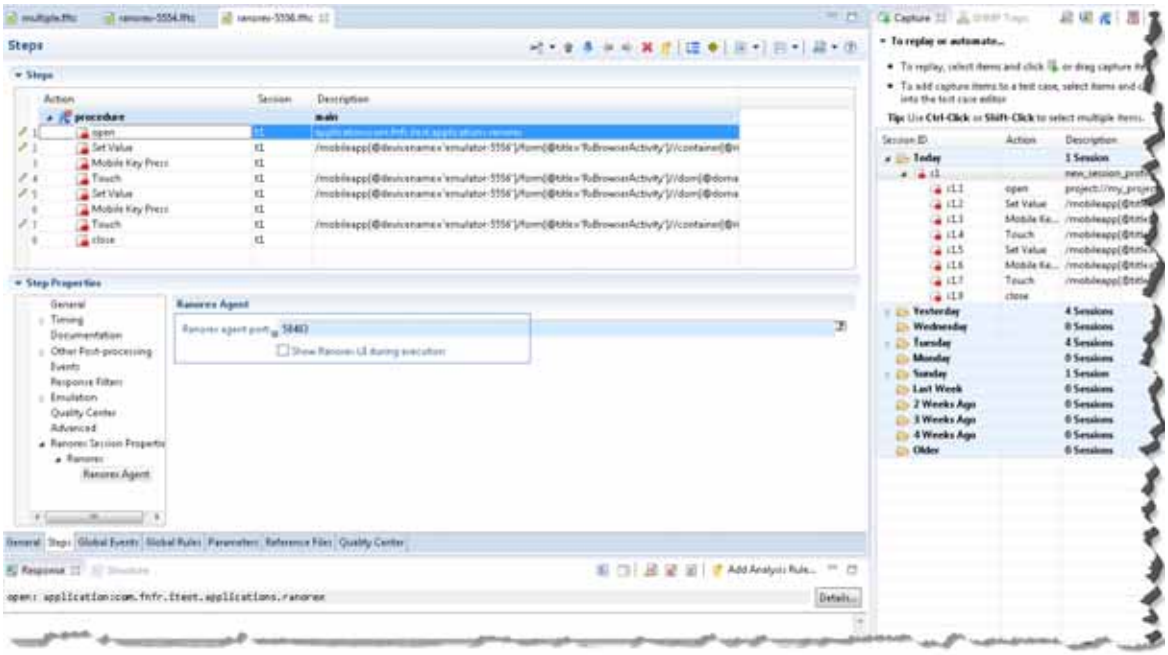
   For this example, generate two different test cases with the elements captured and validated in different sessions.

Note Make sure that the **Ranorex Agent port** for the two test cases are set to execute on different ports and **Show Ranorex UI during execution** is *not* selected.

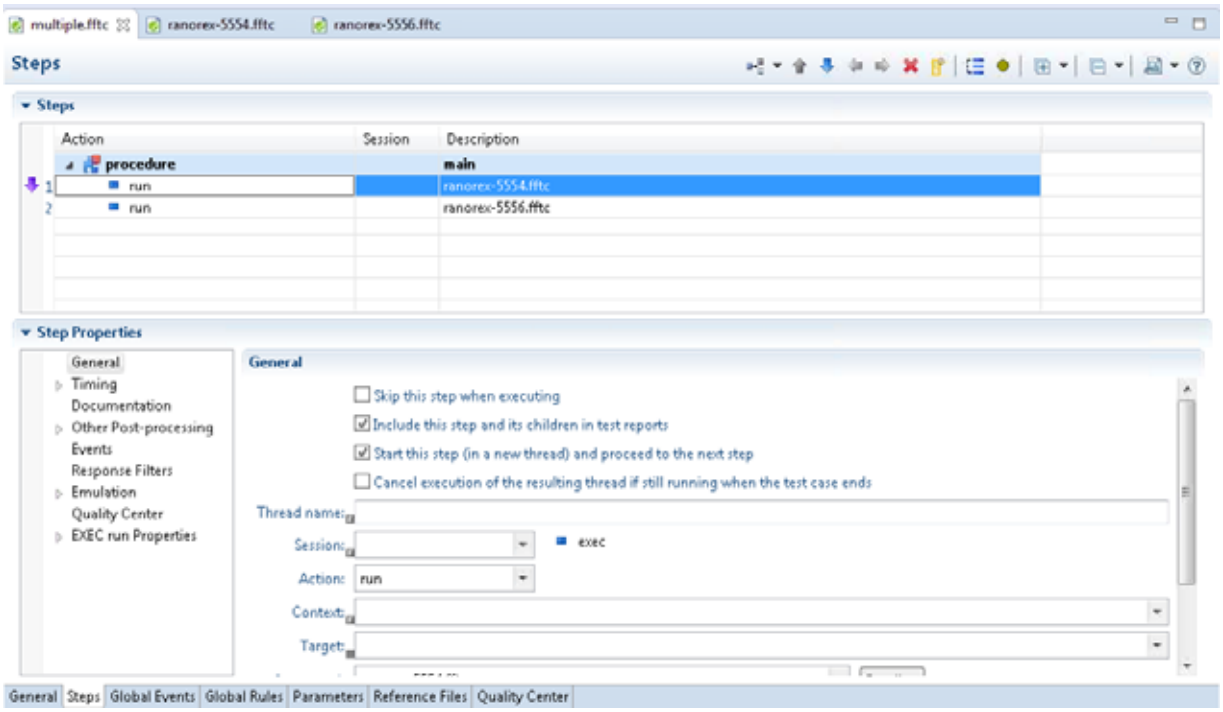
Example 1 (**ranorex-5554.fftc**): Test case generated from the captured and validated elements.



Example 2 (**ranorex-5556.fftc**): Test case generated from the captured and validated elements.



4 Create a test case to run multiple test cases, that is, test cases generated above (illustrated in Example 1 and Example 2.



Note Make sure to configure the **Timing** properties to define step start, duration and finish time to ensure the tests run as required.

5 Click  to **Start Execution in New Window.** The test case starts simultaneous execution as required.

After the test case completes execution, the test results display.

Note You may also set up a single test case to run multiple Ranorex sessions simultaneously when testing mobile devices.

Limitations

- Ranorex requires browser plugin to test DOM elements of web-page (as it is unable to directly test DOM elements of web page). Use Ranorex Instrumentation Wizard to install browser plugin.
- Test cases are not fully multi-browser. For example, if you capture a test case in Firefox browser it is not guaranteed that it will pass in another browser.

Setting preferences for Ranorex

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Session Types > Ranorex**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Ranorex

Ranorex installation path	Enter the installation path (where you installed Ranorex), for example, **C:\Program Files (x86)\Ranorex** **Note** In order for Ranorex to work in iTest, first install Ranorex and then specify the installation path.

CHAPTER 49

EggPlant Sessions

Overview

EggPlant Test session provides ways of automating execution of image-based GUI testing. The seamless testing of these iTest functionalities is achieved by iTest integration with EggPlant application (https://eggplant.io/eggplant-integrations/). The following highlights EggPlant functional within iTest:

- Provides advanced image analysis technology to conduct UI testing, which allows validation the system under test (SUT).
- Helps you test your applications by automating the execution of your functional testing (better, faster, and with less effort).
- Provides functional test automation with (EggPlant's patented) image-based approach to GUI testing.
- Provides a way to interact with any device (including mobile, tablet, desktop, server, and the Internet of Things), capture images, uses image and text search algorithm to locate objects on a screen and write tests.

iTest then allows the saved EggPlant sessions to be injected into iTest test cases for analysis. In addition iTest also supports eggDrive, a mechanism for external processes, such as a script captured from other testing tool, to use EggPlant Functionals automation capabilities.

eggDrive run as a background service and provides a simple XML-RPC interface for iTest (clients) to send messages to EggPlant Functional. The external client sends any standard EggPlant Functional command, and receives the results back. EggPlant Functional retains state between calls, allowing a sequence of message interactions to be viewed as a single session. Drive mode is available on all floating license versions of EggPlant Functional.

Important iTest requires for EggPlant and eggDrive to installed before you can create an EggPlant session or use eggDrive capability.

This chapter includes these topics:

Note Before you begin, see "EggPlant Functional commands" on page 1009 to understand the commands in EggPlant.

General recommendations when running EggPlant test case

Each eggDrive service only supports a single eggDrive session. So iTest does not support multiple EggPlant integration session using the same EggDrive service.

Tip It is recommended to start separate eggDrive service with different ports.

EggPlant Functional commands

The following lists the functional commands available when testing EggPlant sessions.

Note For details, refer to http://docs.testplant.com/ePF/using/epf-qr-eggplant-commands.htm.

Commands	Syntax / Examples	Behavior
Assert	assert x > 10 or assert that x > 10 A statement you want EggPlant Functional to evaluate. To improve readability, you can insert the word ***that*** into an **assert** command as shown in the second example above.	Evaluates the specified statement as true or false, and logs a success, failure, or warning message, or an exception depending on the settings of related properties:
BeginTestCase	BeginTestCase "A"	The BeginTestCase command is used to open a test case, and the EndTestCase command is used to close one. Each test case is given a name, put in quotes after the command.
EndTestCase	EndTestCase "A"	
CaptureScreen	CaptureScreen (Name: "ImageFileName", Rectangle: ((67,33), imagelocation: "OtherCorner"))	Captures a snapshot of the entire Viewer window, or a rectangle indicated in the property list. You can customize this command with a list of any number of the properties. **Tip:** Incrementing is helpful if you are capturing several snapshots within a script (as in a loop or frequently called handler.) The images can all have the same image name, with the incremented number distinguishing them; otherwise, each image would overwrite the previous image with the same name.
Log	Log "Message to be recorded in log file"	Inserts a comment into a script's Log file
LogError	LogError "messageToBeRecordedInLogFile"	Inserts a comment into a script's Log file in red text
LogSuccess	LogSuccess "The page loaded properly!"	Inserts a comment into a script's log file and adds one to the success count for the script.
LogWarning	LogWarning "Message to be recorded in log file"	Inserts a comment into a script's Log file in orange text

Commands	Syntax / Examples	Behavior
CaptureTextImage	CaptureTextImage (text: "some words", rectangle: (0,0)(50,20))	Creates a text image. This command is only used in scripted text-image generators.
KeyDown	KeyDown ControlKey, CommandKey, "d" KeyUp ControlKey, CommandKey, "d" KeyUp AllKeys	Presses and holds the given keys until they are released by a KeyUp command.
KeyUp		Releases keys that have been held by the KeyDown command.
Click	Click "ImageOne", "ImageTwo", "ImageThree" Click (2,4)	Clicks the SUT mouse in a coordinate location, or in the first image parameter found
Connect	Connect “192.168.1.110” Connect (serverID: "10.211.55.4", portNum: "5900", username: "harris", password: "slayerfan", Visible: "Yes") Connect (ServerID: "Samsung_S5", AdditionalArgs: ("--scale-screen 2")) --For Android SUT connections	Opens a VNC or RDP connection with a SUT; makes it the active connection.
Disconnect	Disconnect (serverID:"192.168.1.20", port:"5900") Disconnect "Nightly Regression" Disconnect	Closes a connection with a SUT
Copy File	copy file "/local/path/to/file" to "SUT:/remote/path/to/file" -- send file to the SUT copy file "sut:/remote/requested/file" to "/local/path/to/receive/file" -- get a file from the SUT	This command lets you copy a file either from the local machine to the SUT or from the SUT to the local machine. Use the prefix SUT : before the path to indicate the SUT. Note that you can't use this command to copy a file from one location on the SUT to a different location on the SUT or to a different SUT. Copy File works only with single files, not with folders.
DoubleClick	DoubleClick "SomeImage", (Text:"SomeWords",Italic: No) DoubleClick (25,358)	Double-clicks the SUT mouse in a coordinate location, or in the first image parameter found.
DoubleTap	DoubleTap (320,504)	This command executes a double tap at either the current location or the location specified by the parameter.
Drag	Drag "ImageOne", "ImageTwo" Drag (231, 100)	Clicks and holds the SUT mouse in the Hot Spot of the first location found
DragAndDrop	DragAndDrop "ImageOne", "ImageTwo", (980, 322)	Clicks and holds the SUT mouse in the first location, moves to subsequent locations, releasing the mouse button on the last location
Drop	Drop "ImageOne", Drop (231, 100)	Moves the mouse to the given location and releases the mouse button

Commands	Syntax / Examples	Behavior
ExecuteRemoteCommand()	ExecuteRemoteCommand("mkdir /data/local/tmp/TestDir") -- on Android, this command creates a directory on the device using shell command syntax ExecuteRemoteCommand("UIATarget.localTarget().systemName()") -- on iOS, this command gets the system name of the device with the active connection. put ExecuteRemoteCommand("ls /data/local/tmp/; echo hello", waitFor:20) --You can pass multiple commands by separating them with semi-colons as shown in this example	for running commands on SUTs that are mobile devices. You can use it to return data or not. The command is either synchronous (waits for a response) or asynchronous (does not wait). The way in which it runs commands differs with the mobile operating systems as follows: On Android, `ExecuteRemoteCommand` runs as a shell command on the actual phone. On iOS, `ExecuteRemoteCommand` runs the command as JavaScript, making calls to the Apple UIAutomation API. **Note:** Using the **ExecuteRemoteCommand()** command requires iOS Gateway 2.6 or the embedded Android VNC server in eggPlant Functional 15.21.
InstallApp	**Note:** This command is for use with the iOS Gateway only. **Example:** `installApp "/Path/to/App/AppName.ipa"` installApp (applicationPath:"/Path/to/App/AppName.ipa") installApp (applicationPath:"/Path/to/App/AppName.ipa", provisionPath:"sut:/Path/to/Provisioning/Profile/YourProvisioningProfile .mobileprovision", certificateName:"your iPhone Developer certificate name")	This command installs an app to the mobile device. You specify the app to install with the applicationPath parameter. Optionally, the command instruct iOS Gateway to developer sign the app before installing it if you include the provisionPath and certificateName properties. Applications to be installed can be specified with local paths if they are on the machine on which you are running eggPlant Functional; you can also install apps that are already on the machine where you're running iOS Gateway by using the `SUT:` prefix on the path. This command waits until it receives a completed message from the server before proceeding. **Note:** Your signing certificate for the certificateName property must be installed on the iOS Gateway machine before using it with `InstallApp`.
KillApp()	KillApp("sampleApp") -- Kills a developer-signed app named "sampleApp." KillApp("com.google.android.apps.maps") -- Kills the Maps app on Android.	Terminates the specified app.
LaunchApp	LaunchApp "MyApp" LaunchApp "Springboard" LaunchApp "MyiPad : MyApp"	Launches a signed app (or Springboard) on an iOS device that you are connected to through the iOS Gateway.
MouseButtonDown	MouseButtonDown 2 MouseButtonUp 4	Presses the SUT mouse button
MouseButtonUp	MouseButtonDown 2 MouseButtonUp 4	Releases the SUT mouse button

Commands	Syntax / Examples	Behavior
MoveTo	MoveTo "ImageOne" "ImageTwo" MoveTo (190,87)	Moves the SUT mouse to a coordinate location, or to the first image parameter found
MoveToEach	MoveToEach "ImageOne", (780, 91), "ImageThree"	Moves the SUT mouse to each of the given coordinate locations and image parameters
PauseScript	PauseScript	Pauses script execution; enters debug mode
PinchIn	PinchIn PinchIn (At:(500, 500), Duration:5.0)	The PinchIn command acts like a two-point pinch on a screen; the effect of pinching in is to zoom out (i.e., the screen image effectively gets smaller). For the PinchIn command, the At point is static or "fixed"; the From point moves toward the At point. **Note:** Not all screen views or apps support zoom in or out behavior. For instance, using a pinch command while a device is on the Home screen has no effect.
PinchOut	PinchOut PinchOut (Distance:300)	The PinchIn command acts like a two-point pinch on a screen; the effect of pinching out is to zoom in (i.e., the screen image effectively gets bigger). For the PinchOut command, the At point is static or "fixed"; the To point moves away from At point. **Note:** Not all screen views or apps support zoom in or out behavior. For instance, using a pinch command while a device is on the Home screen has no effect.
Press	Press "AppIcon"	This command executes a tap and hold at either the current location or the point specified by the parameter.
PressBackButton	PressBackButton	Pushes the Back button on Android devices running Android 4.0.4+..
PressHomeButton	PressHomeButton	Pushes the Home button on iOS devices, as well as Android devices running Android 4.0.4+..

Commands	Syntax / Examples	Behavior
Reboot	When you use the `Reboot` command in your script, we recommend sending a `disconnect` command immediately after it, and then reconnecting. When you reconnect, allow time for the device to come up. For example, you can include the `connect` command in a repeat command that tries the connection until it succeeds as shown in the example below: Connect "<sut_name>", port:5900 Reboot Disconnect Repeat until the connectioninfo.connected is true try to connect "<sut_name>", port:5900 Log the counter End repeat	The **Reboot** command is for mobile devices. This command restarts the mobile device with the active connection. **Note:** Using the **reboot** command requires iOS Gateway 2.6 or the embedded Android VNC server in eggPlant Functional 15.21.
Release	Release "AppIcon"	This command releases a Press action at either the current location or the point specified. It is the equivalent of a MouseButtonUp command, catered to mobile devices.
RotateLeft	RotateLeft (WaitFor:10.0)	The RotateLeft command rotates the mobile device screen one quarter turn (90 degrees) clockwise from the current device orientation.
RotateRight	RotateRight	The RotateLeft command rotates the mobile device screen one quarter turn (90 degrees) counter-clockwise from its current device orientation.
RefreshScreen	RefreshScreen	Redraws the image of the SUT in the Viewer window
RightClick	RightClick (imageName: "SomeImage", searchType: Text) RightClick (12,355)	Right-clicks the SUT mouse in a coordinate location, or in the first image parameter found
Run	Run "OtherScript", "parameter1", "parameter2"	Runs a given script
RunWithNewResults	RunWithNewResults "SomeScript", "parameter", "parameter2"	Runs a given script, generating a separate Log file for it
SetDeviceOrientation	SetDeviceOrientation LandscapeRight	This command changes the device orientation to what you specify with the parameter. It waits until the new orientation is achieved before proceeding; however, you can include a WaitFor delay as well to control timing issues.
ScrollWheelDown	ScrollWheelDown 5 ScrollWheelUp 10	Scrolls the SUT mouse wheel down a given number of increments

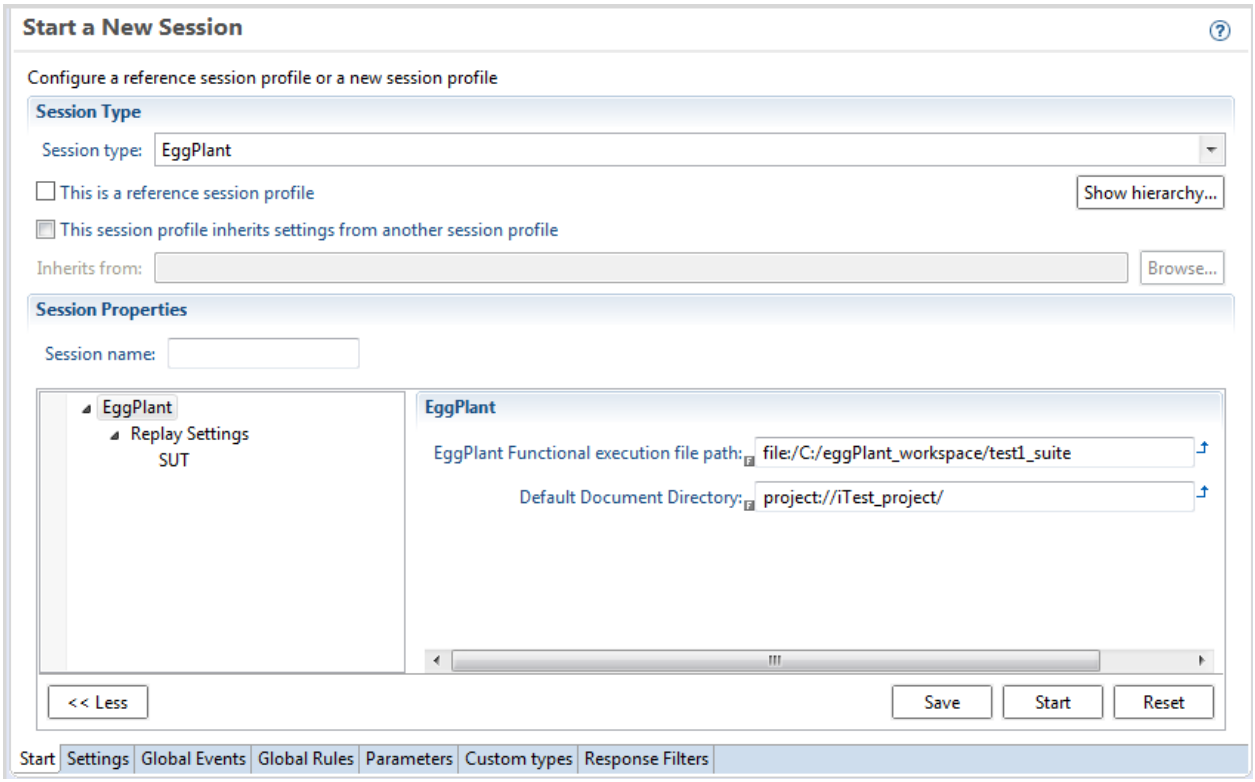
Commands	Syntax / Examples	Behavior
ScrollWheelUp	ScrollWheelDown 5 ScrollWheelUp 10	Scrolls the SUT mouse wheel up a given number of increments
SendMail	sendmail (to: "testmanager@somecompany.com,tester1@somecompany.com", subject: "Test failed", body: "The very important test script generated an error. The log file is attached.", attachment: logfile)	Sends an e-mail from within a script
SetOption	SetOption scriptlogging, yes SetOptions (>searchrectangle: (2,4,100,4), scriptlogging: yes) SetOptions JoesOptions	Modifies a global property value
SetOptions	SetOption scriptlogging, yes SetOptions (>searchrectangle: (2,4,100,4), scriptlogging: yes) SetOptions JoesOptions	Modifies multiple global property values
SetRemoteClipboard	SetRemoteClipboard "new clipboard contents"	Places text in the SUT clipboard
ShowRemoteWindow	ShowRemoteWindow	Shows the Viewer window
SwipeDown	SwipeDown (320,840)	This command executes a vertical swipe motion, starting from either a central location at the top of the screen or from the point indicated by the parameter, and moving downward. This motion imitates a hand-executed swipe on a mobile device by moving in an arced fashion.
SwipeLeft	SwipeLeft (600,400)	This command executes a horizontal swipe motion, starting from either a central location on the right side of the screen or the point indicated by the parameter, and moving to the left. This motion imitates a hand-executed swipe on a mobile device by moving in an arced fashion.
SwipeRight	SwipeRight "LeftImage"	This command executes a horizontal swipe motion, starting from either a central location on the left side of the screen or the point indicated by the parameter, and moving to the right. This motion imitates a hand-executed swipe on a mobile device by moving in an arced fashion.
SwipeUp	SwipeUp "StartImage"	This command executes a vertical swipe motion, starting from either a central location at the bottom of the screen or from the point indicated by the parameter, and moving upward. This motion imitates a hand-executed swipe on a mobile device by moving in an arced fashion.

Commands	Syntax / Examples	Behavior
Tap	Tap "Safari_Icon"	This command executes a tap at either the current location or the location specified by the parameter.
Hide RemoteWindow	Hide RemoteWindow	Hides the Viewer window
HighlightRectangle	HighlightRectangle ((FoundImageLocation(), FoundImageLocation() + (50,10)) HighlightRectangle ((FoundImageLocation(), FoundImageLocation() + (50,10), white) HighlightRectangle (10,10,50,50), (Duration: 1 minute)	Draws a border around the given rectangle in the Viewer window
Hide RunWindow	Hide RunWindow	Hides or shows the Run window. Tip: You can use the Hide command to close the Run window when you do not need to watch a script run, and insert a Show command to reveal the Run window for the parts of the Run that you want to see.
Show RunWindow	ShowRunWindow	Shows the Run window. Tip: You can use the Hide command to close the Run window when you do not need to watch a script run, and insert a Show command to reveal the Run window for the parts of the Run that you want to see.
SetSearchRectangle	SetSearchRectangle ((0,0),(100,100))	A search rectangle can be specified with exact coordinates, but it is more common in EggPlant Functional to use images to define the rectangle, in case the location of the text is not always the same. This might happen, for instance, if the window it is in doesn't appear in the same location on the screen every time the test is run. Note: When using images to define the boundaries of the search rectangle, keep in mind that the points used to set the rectangle are based on the hot spot, which can be moved outside of the image.
TraceScreenOn / TraceScreenOff	TraceScreen On TraceScreen Off	Turns screen captures before every command on and off.
TypeText	TypeText "Some Words" TypeText <<Some Words with a Return.>> TypeText AltKey,"x"	Sends keystrokes to the SUT keyboard
Wait	Wait 6 Wait 10 minutes	Delays the next line of execution for the given length of time

Commands	Syntax / Examples	Behavior
WaitFor	WaitFor 6.5, "ImageOne", "ImageTwo" WaitFor 2 minutes, "SomeImage"	Delays execution of the next line until any one of the given images is found
WaitForAll	WaitForAll 6.5, "SomeImage", "SomeImageFolder" WaitForAll 2 minutes, "SomeImage", "SomeImageFolder", "SomeImage2"	Delays execution of the next line until all given images are found

Creating EggPlant Session Profile in iTest

- Create an EggPlant session profile.
- Select the **EggPlant** session from the session type list. Select:
  - **EggPlant functional IDE path**: Browse and select the EggPlant installed.
  - **Default Document Directory**: Select the location for the information captured via EggPlant to be copied.



Note Use the EggPlant file as below. For example:

- For Windows:
  - If EggPlant Functional IDE version is newer than 16.0.1 use the following: **file:/C:/Program%20Files/eggPlant/eggPlant.bat**
  - If EggPlant Functional IDE versions is older than 16.0.1, use the following: **file:/C:/Program%20Files%20(x86)/eggPlant/eggPlant.exe.**

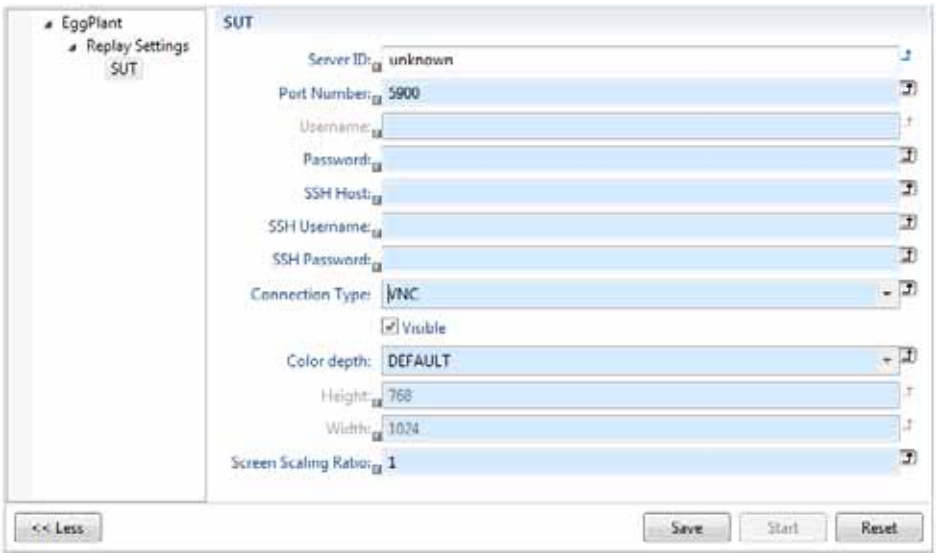
- For Linux: file:/usr/local/bin/runscript

  For Linux, provide the URI to execute runscript file located in the same folder as EggPlant file.

- Click **More** and enter the **Replay** and **SUT** settings.

  In **Replay Settings** window make sure to enter the correct test **Suite Path** and the **Drive Port** (this is to run the eggDrive installed, if it is not running on this drive port). If required, select **Capture screen before executing a step** to capture a screen shot before executing a step.

  In **SUT** window, enter the appropriate Server ID and Port Number and other required settings.



- Save the session and click **Start** to launch EggPlant.

- Perform the required activities using Eggplant Functional as described in "Using EggPlant Functional" on page 1017.
- Close EggPlant Functional, Convert the latest EggPlant sessions and inject it into iTest capture service.
- Generate new test case from captured session and run this test case. See "Generate iTest test case from captured EggPlant session" on page 1024.

Using EggPlant Functional

Follow these steps to interact with EggPlant session to capture user interaction and create a session that can be included in iTest.

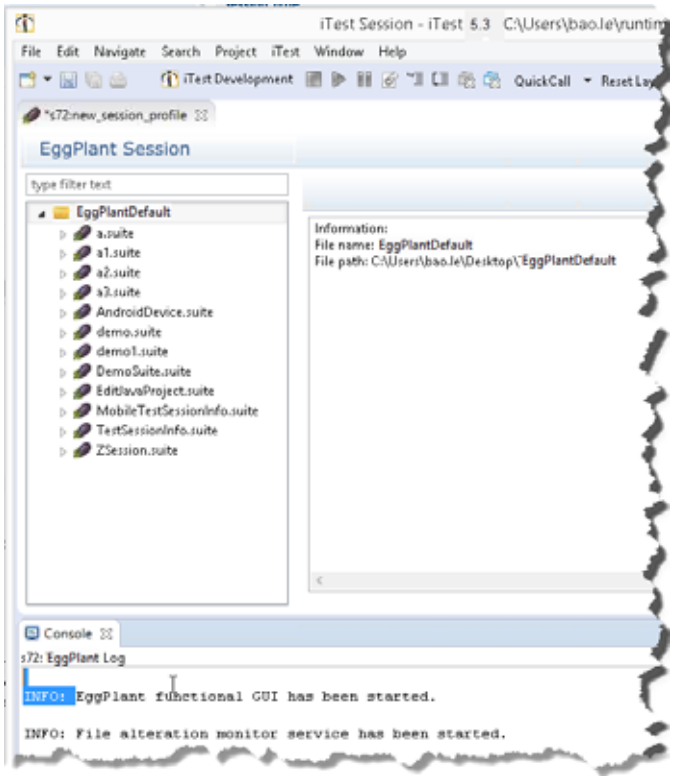
Step 1 Open EggPlant session

Open a EggPlant session, for example, session created as described in "Creating EggPlant Session Profile in iTest" on page 1016.

Step 2 Start EggPlant Session

Click the **Start** button to run EggPlant session.

- The EggPlant session editor in iTest starts with the ROOT_URL pointing to the default workspace specified.



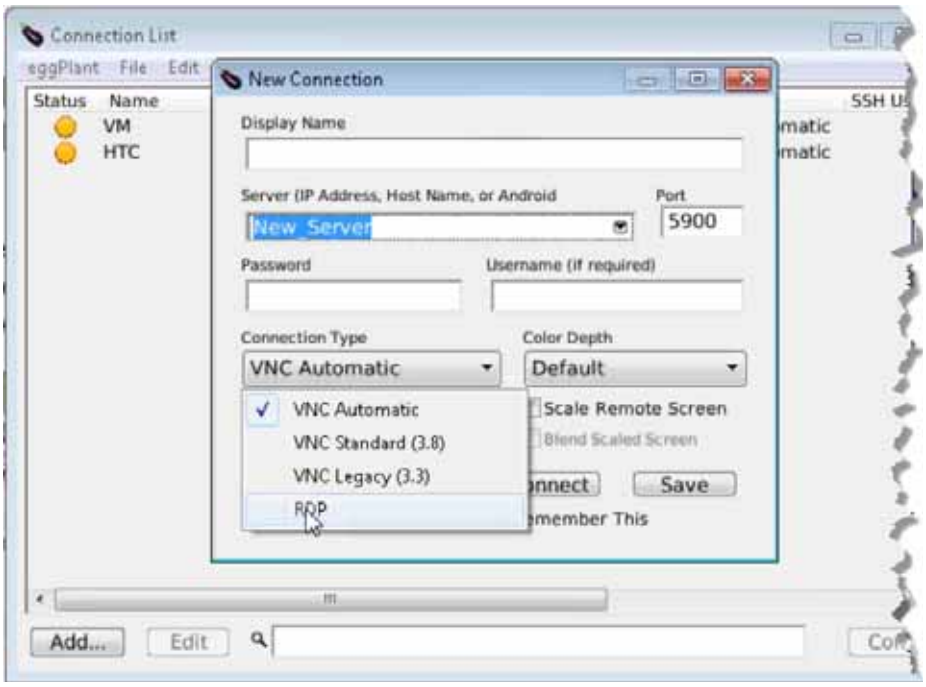
- The EggPlant function launches with the default workspace as set in the session profile:

  Default Document Directory location

Note It is recommended that you only create/open suite under the specified default workspace in **Default Document Directory**.

Step 3 Create Connection

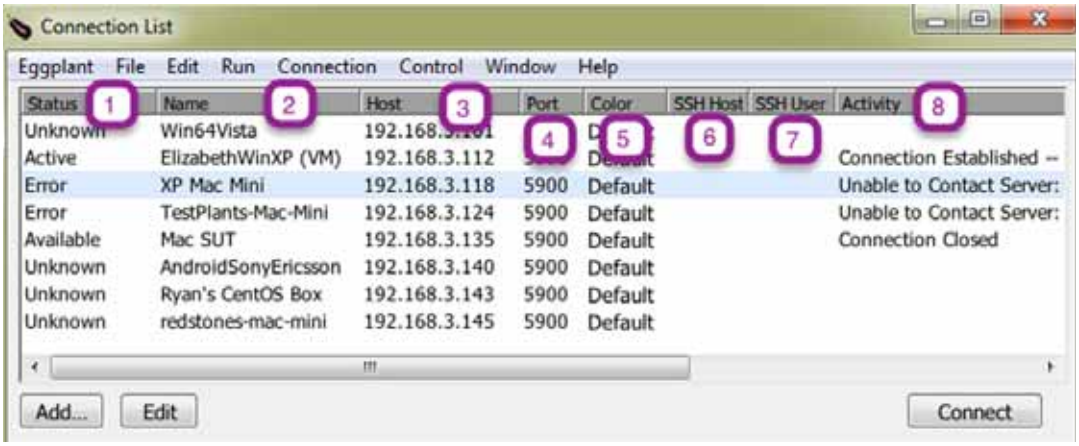
You Create a connection to a SUT from EggPlant Functional by setting up a new connection or connecting to an existing connection. For new connection, enter the **Display Name** and **Port**.



The **Remember This Connection** option is enabled by default. When enabled, EggPlant Functional saves this connection in the Connection List for use in the current and future EggPlant Functional sessions. When disabled, the EggPlant Functional creates the connection as a temporary connection for the current session only and will not be available in future sessions. Temporary connections display with their names in italics in the Connection List.

Important When **Remember This Connection** option is disabled, iTest cannot capture the connection information for the captured open step.

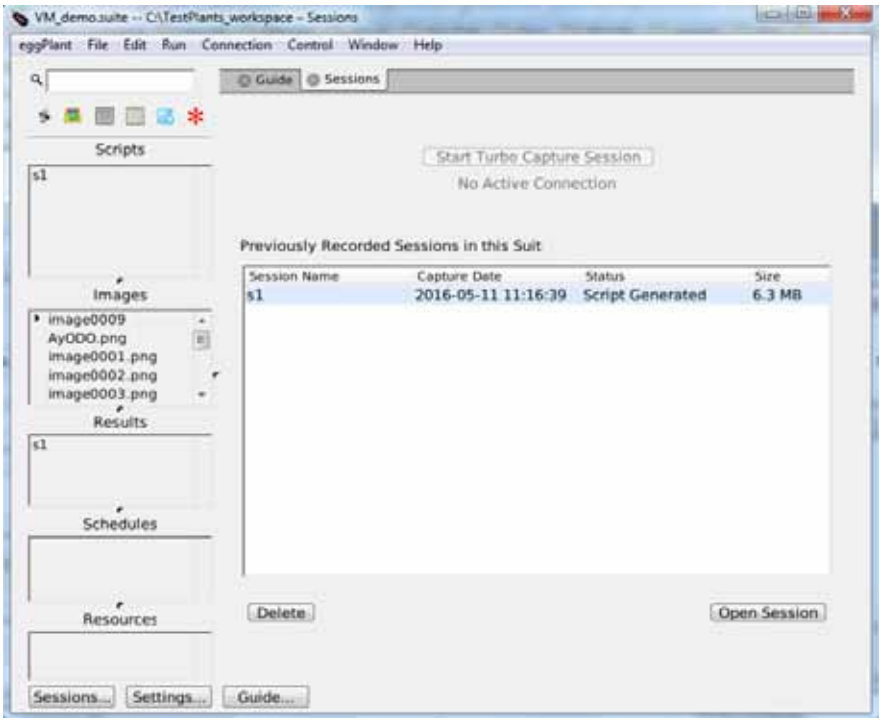
For example, the diagram below illustrates a window that allows creation of VNC connections to various systems under test (SUTs), whether via a direct connection over Wi-Fi or USB, or a relayed connection through eggCloud or the iOS Gateway.



Step 4 Start a Turbo Capture Session

Start Turbo Capture session, for example: (See http://docs.testplant.com/ePF/using/epf-using-turbo-capture.htm for details).

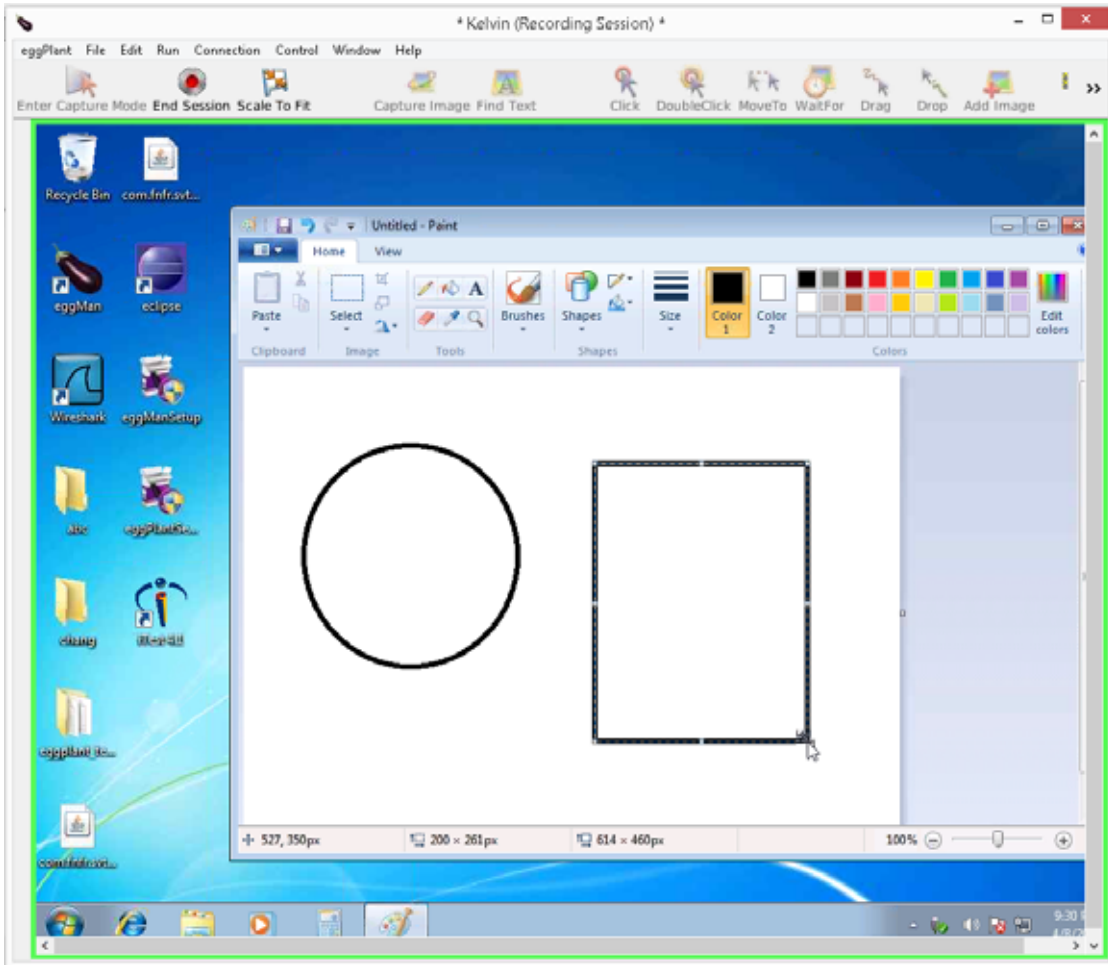
- Begin a session through the Suite window. .



- Click the **Sessions** button in the lower left corner of an open Suite window.
- Click the **Start Turbo Capture** button in the open Sessions tab.
- Begin a session through the Viewer window.
- In the Viewer window, click the Start Session icon.

  The Viewer window allows you to see the SUT to which you are currently connected. This is indicated by the Viewer window being outlined with a green rectangle, and the End Session button being visible in the toolbar, as shown below.

Note You can have multiple Viewer windows open at once, if you have multiple connections to different SUTs. The toolbar icons at the top of the window are used as part of the assisted scripting functionality.



- All actions such as clicks, scroll wheel movements, and keystrokes are recorded automatically. To capture mouse movement, hold the Shift key down while moving the mouse. When the Shift key is released, mouse movements will no longer be recorded, and the session will go back to recording click, scroll wheel, and keyboard events only.
- During session creation, screen captures are taken along with each action that is executed, and these screenshots will be used to create images in the process.

Step 5 Stop the Turbo Capture Session and Customize captured steps

Close the **EggPlant Functional** GUI and then close the **EggPlant session editor**, once all of your images have been sized and named, and all TypeText commands have been modified as needed.

See EggPlant documentation for details.

Step 6 Open the Captured Session, modify, and Generate Script

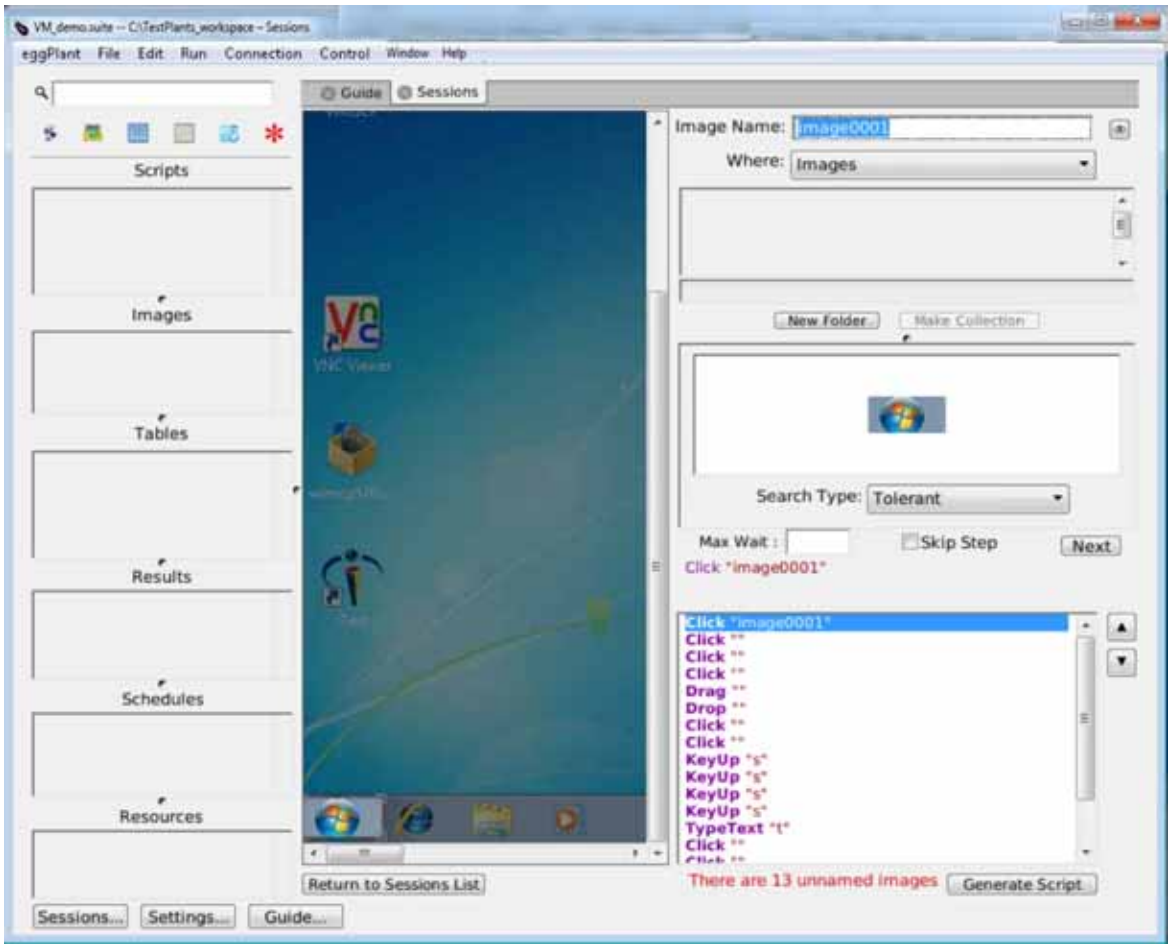
Open the Captured session, modify and generate script in EggPlant Functional UI. Select the session in EggPlant functional GUI and modify the captured steps, for example, name, maximum wait, and so on. Select your session and click Open Session.

Note This will put the Sessions tab in script generation mode.

- The session opens in the Sessions tab.

  On the left hand side, the images from your recorded session will be displayed. If the first action you took interacted with the UI, it will show an approximate Capture Area for the image to be used in the first line of your script.

- On the right hand side, a file browser displays with a field ready to accept the name of your image. The image preview will be displayed below the image browser, along with the script.



- Modify the Capture Area for the image in the view on the left hand side of the session view, using best image capture practices.

  If you are working with a TypeText command, the screen of the SUT will show on the left hand side as it appeared at the time the typing was executed, but no capture area will be proposed.

  Name the image on the right hand side and adjust the search settings and timing as desired.

  TypeText commands will bring up a different editing view, in which you can modify the keys that the script will be typing, or approve them as they are.

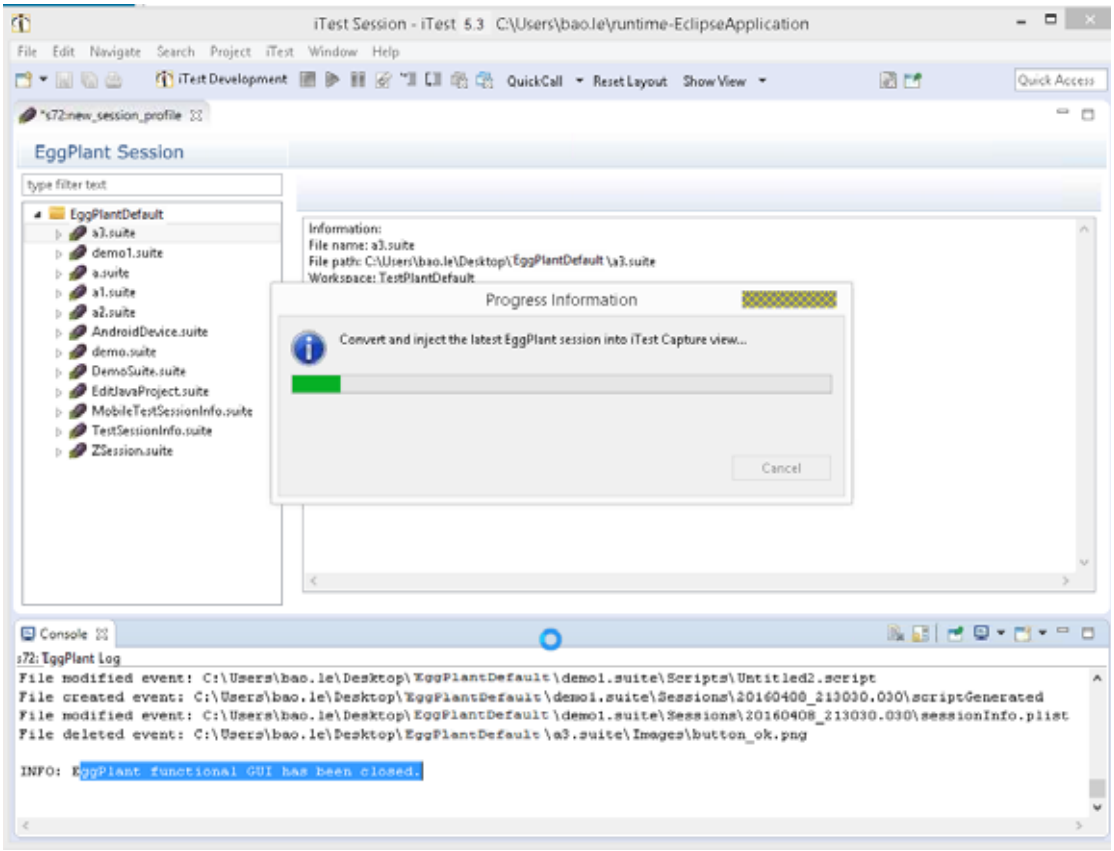
  Press the Return key on your keyboard or click Next after each change to move on to the next line of code.

Tip When you click/press the Return key on your keyboard or click Next after each change, the session's current state is saved, so you can leave the session to do other work, and come back to finish it later.

Note The EggPlant Console Log logs all file change events, information, and errors.

Step 7 Inject EggPlant session into iTest Capture View

The EggPlant sessions are automatically converted and inject it into iTest capture when you close EggPlant Functional GUI.



Note iTest supports capturing only the latest session and does not support capturing multiple EggPlant functional sessions. That is, only the latest EggPlant session is converted and injected into the iTest capture view, even if have updated or created multiple sessions in EggPlant.

iTest automatically injects CaptureScreen commands in between the converted iTest test steps.

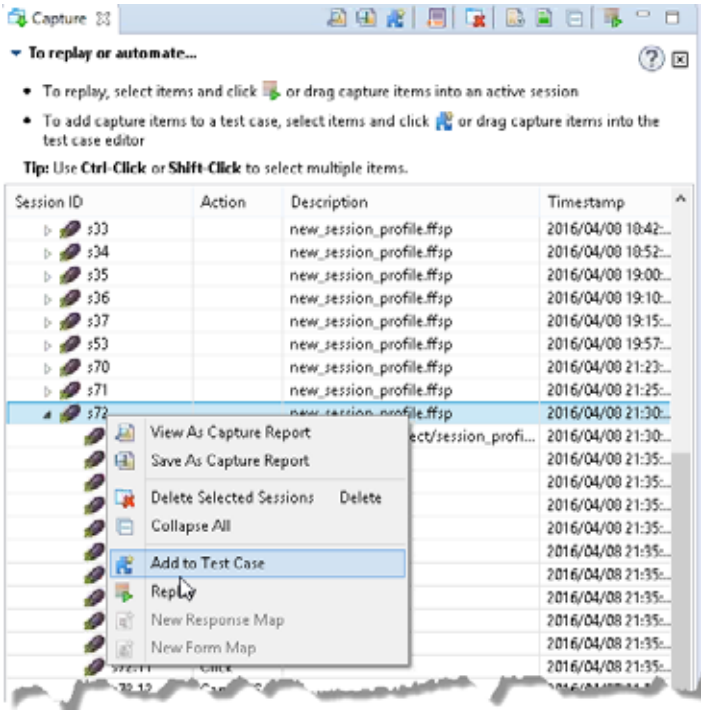
See section "Generate iTest test case from captured EggPlant session" below.

Generate iTest test case from captured EggPlant session

Perform these tasks in iTest Test Case editor window after capturing EggPlant session.

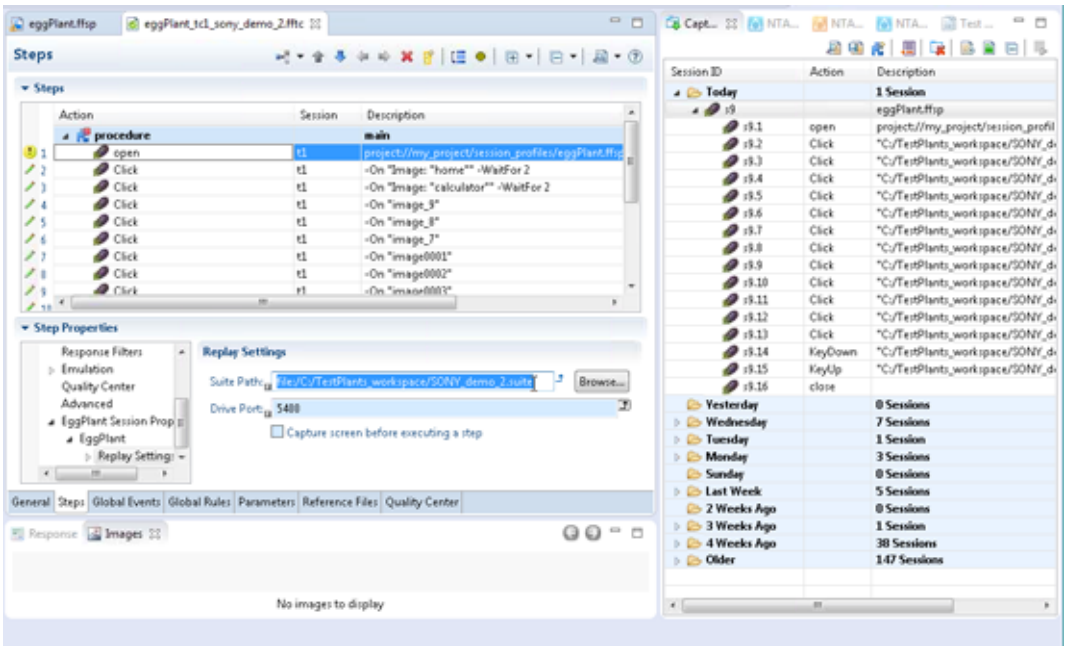
- In the iTest Session window, open/select, Show View/Capture. This will display all your recorded sessions and images on EggPlant Functional.
- Select the EggPlant session to be used to generate an new Test Case, right-click and select Add to new test case.

Note The example below shows only the captured session with their steps..



- The **Add to Test Case Wizard** opens, select **Create a new test case using the captured items**, enter a new test case name and click **Finish**.

- A new test case generates from the selected captured session, which includes the steps required to execute the test case.



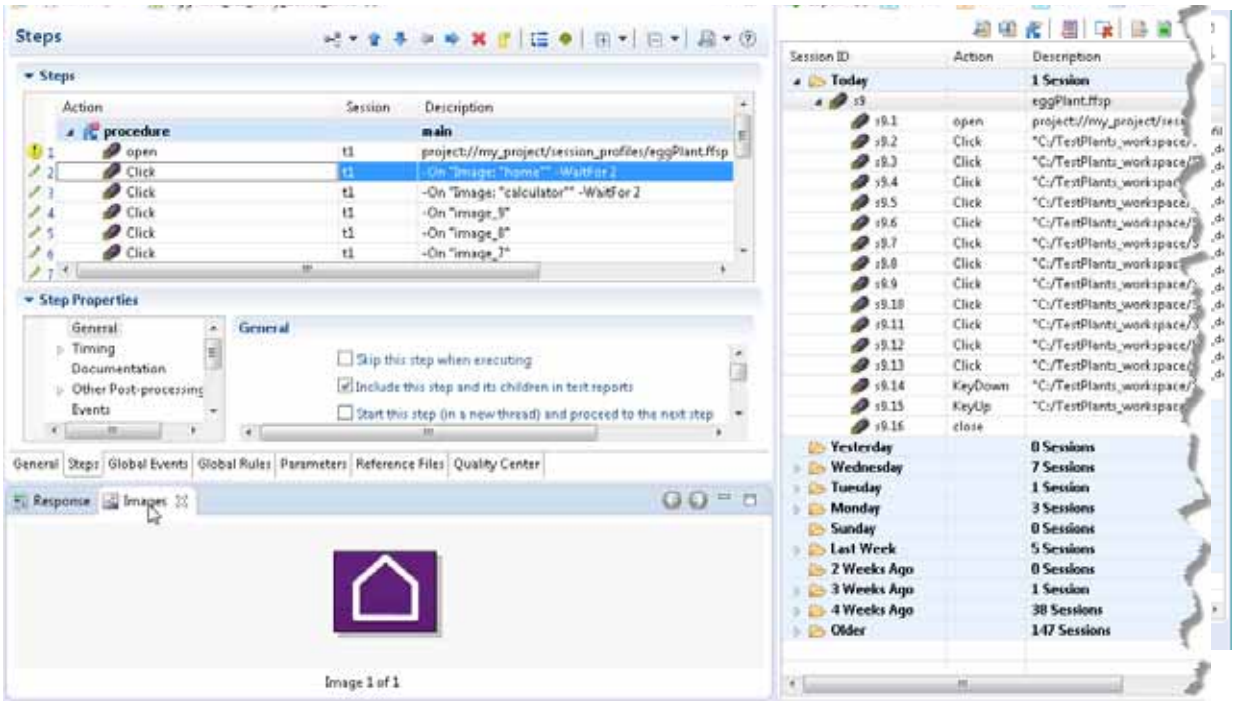
- Configure the eggDrive port service. See **Replay Settings** on 1017 from session profile or open step properties. Refer to http://docs.testplant.com/ePF/using/epf-using-eggdrive-with-eggplant-functional.htm for details.

Note The eggDrive will start automatically based on the port number specified in session profile, if eggDrive is not already running.

Important The Suite Path properties on "Replay Settings" page and Server ID property on SUT page under the open step session properties page can be overridden by the information provided during starting session.

Replay

After running this test case, you may Show **Image** view to see the captured image or Show **Response** view to see the response for each step.



Note You may also select **Capture screen before executing a step** to capture a screen shot before executing a step (see page 1017)

Modify iTest EggPlant test case

Perform these tasks in iTest Test Case Editor to modify the test case generated (see "Generate iTest test case from captured EggPlant session" on page 1024) using the captured EggPlant session. For example, to add more images to the EggPlant test case previously generated.

To add more images to an existing EggPlant test case:

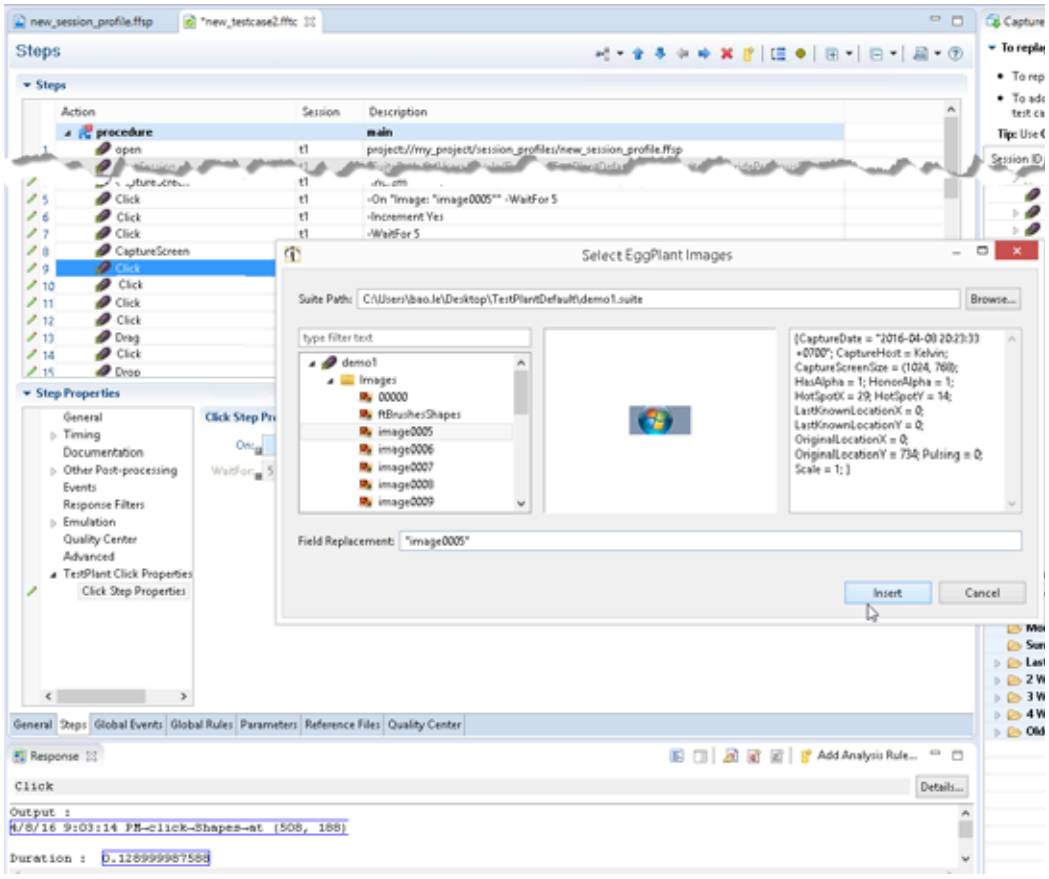
- Go to Click Step Properties, right-click select **Insert >EggPlant Image.**

  By default select and insert an EggPlant image follow images in the test suite path specified from **Open** step opens.

- Select/click a single image to view and see the image details.
- Click Insert to add the image into the **Click Step properties > On:** parameter.

Note You may also select multiple images to add Or select a folder, which indicates that you have selected all images under this folder.

- If a test suite path is not detected, from the **Open** step, a **Browse** button becomes available for you to navigate to the appropriate suite path to select the images.



CHAPTER 50

ADB Custom Session

ADB custom session overview

The ADB (Android Debug Bridge) custom session allows you to communicate with an emulator instance or connected Android-powered device. The steps below lists the prerequisites to building and running iTest ADB custom session.

- Build ADB custom session type using Session Builder, based on SSH sessions.

  See Chapter 35, “Session Builder” on page 808 (and video to understand how to use Session Builder) and also Chapter 81, “SSH Sessions” on page 1522.

- Make sure that the device is unlocked to enable the use of ADB commands (for example, copy file to device, install new app on the device, or access device database, system information, and etc.)
- Enable USB debugging (see Enabling ADB Debug for details).
- Make sure to specify the device serial number in command parameters if more than one device or emulator are connected. If device serial numbers are not provided in command parameters, an error displays saying that more than one device is connected to the PC at the same time.

  Each device connected to PC via USB port will have a device serial/ID. The device serial is unique and is used to distinguish each device when more than one device is connected to the PC at the same time.

Note The ADB custom session type supports Android versions: 4.4kitkat, lollipop 5.0.

ADB Session

In iTest, open a session, select ADB, and specify the required parameter values.

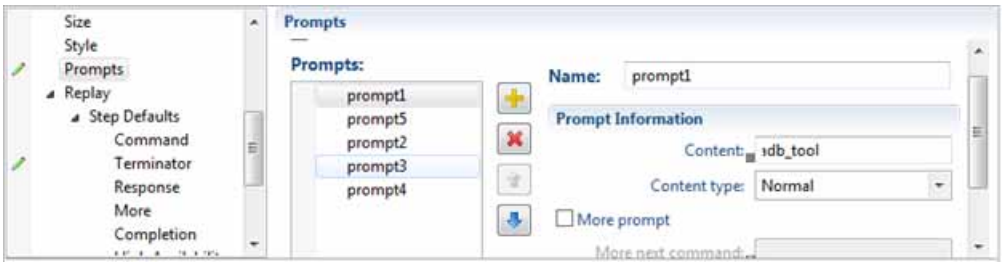
Note As ADB custom session is based on SSH, it is mandatory to specify appropriate value in the **Prompts** section of the Session profile. The session will hang if no value is specified in the **Prompts** section.

The most common prompts in ADB session are:

- ADB tool directory
- Initial terminal shell directory (SSH). for e.g.: **C:\Users\user1**
- Devices terminal shell. e.g.: **shell@htc_eyetuhl:/ $**

- Devices terminal shell login as administrator (su), e.g., **su@htc_eyetuhl:/ $**

  For example, add prompt **Regex**: for Linux: **.*\$** and for Windows: **.*>**



ADB custom session command set

This topic lists the ADB custom commands that you can submit from iTest.

Important Make sure to run the initialize handset command (**initialize_handset_adb**) before running any other command. This will initialize the environment, any installed 3rd party apps, check for device status, etc.
You must specify **adb_dir** (ADB tool directory) and **android_tool_dir** (folder containing 3rd party apps). See the Commands list below.

Command	Description	Arguments	Return Value
initialize_handset_adb	**Important:** It is mandatory to run this command at the beginning. • Initialize the ADB environment. • Install 3rd party tools on devices. **Note** Devices will be restarted during initialization if chompSMS does not exist or if it is not a default SMS app.	**device_serial_list:** list of devices to initialize. If argument is not provided, all the connected devices will be initialized. **adb_dir:** ADB tool directory **android_tool_dir:** Android tool directory which includes busybox, iperf, sqlite3, tcpdump, chompSMS. These tools will be copied to `/data/itest_tools of Androiddevice` by default. These tools are placed in android_tools directory under ADB_session project. **device_tool_dir:** the folder to which the 3rd party app will be copied. (The folder must have r/w permission).	**Success:** "device ID" - device initialize successful. **Fail:** "device ID" - device initialize failed. **Unrooted:** "device ID" - device are unrooted. Response output: Success: FA4BBY800028 Success: FA4BBY800029 Unrooted: FA4BBY800030 Fail: FA4BBY800031 Success: FA4BBY800032 **Third party app:** • chompSMS7.08.apk, • iperf, sqlite3, tcpdump, • busybox
check_network_ connectivity	Using handset to ping a targeted IP address, to test the connectivity of the user plane	**device_serial:** ADB serial number to identify the device i**p_address:** The target address that you would like to check connection	PING localhost (127.0.0.1) 56(84) bytes of data. 64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.169 ms --- localhost ping statistics --- 1 packets transmitted, 1 received, 0% packet loss, time 0ms rtt min/avg/max/mdev = 0.169/0.169/0.169/0.000 ms

Command	Description	Arguments	Return Value
generate_user_plane_traffic	Using Iperf to ping a targeted Iperf server, to generate PS TCP traffic **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device **iperf_server:** iPerf server **iperf_options:** Additional iPerf options	iperf_server: iperf.he.net Response output: WARNING: generic atexit() called from legacy shared library ---------- Client connecting to iperf.he.net, TCP port 5001 TCP window size: 512 KByte (default) ---------- [4] local 10.20.16.81 port 44567 connected with 216.218.227.10 port 5001 [ID] Interval Transfer Bandwidth [4] 0.0-10.2 sec 384 KBytes 308 Kbits/sec **Third party app:** iperf
generate_ping_traffic	Using Handset to ping a targeted IP address, to generate PS ICMP traffic	**device_serial:** ADB serial number to identify the device **ip_address:** The target you would like to ping **count:** Number of echo requests to send **addition_ping_options:** Additional options for ping command	PING google.com (216.58.221.142) 128(156) bytes of data. 72 bytes from hkg07s02-in-f142.1e100.net (216.58.221.142): icmp_seq=1 ttl=50 (truncated) 72 bytes from hkg07s02-in-f142.1e100.net (216.58.221.142): icmp_seq=2 ttl=50 (truncated) 72 bytes from hkg07s02-in-f142.1e100.net (216.58.221.142): icmp_seq=3 ttl=50 (truncated) 72 bytes from hkg07s02-in-f142.1e100.net (216.58.221.142): icmp_seq=4 ttl=50 (truncated) --- google.com ping statistics --- 4 packets transmitted, 4 received, 0% packet loss, time 3003ms rtt min/avg/max/mdev = 39.346/56.152/64.215/9.874 ms
ftp_download	Connect to a FTP server to download files. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device • username • password **port:** Ftp server port **host:** Ftp server host **local_file:** the URI filename to save. **remote_file:** URI of the file on server for download	Return "retcode: TRUE" if download is successful otherwise error messages will be returned. **Third party app:** busybox

Command	Description	Arguments	Return Value
ftp_upload	Connect to a FTP server to upload files **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device • username • password **port:** Ftp server port **host:** Ftp server host **remote_file:** URI of the remote file (target URI of the server where the file is will be uploaded) **local_file:** URI of local file for upload	Return "retcode: TRUE" if download successful otherwise error messages will be returned. **Third party app:** busybox
tcp_dump	Capture handset network packets with tcpdump tool **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device **count:** Exit after receiving count packets **addition_tcpdump_options:** Additional Tcldump options	tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes 15:12:57.167494 STP 802.1d, Config, Flags [none], bridge-id 8010.00:21:1b:c7:c6:00.8005, length 42 15:12:57.985654 IP 10.20.16.76.17500 > 255.255.255.255.17500: UDP, length 247 15:12:57.985849 IP 10.20.16.76.17500 > 255.255.255.255.17500: UDP, length 247 15:12:57.985970 IP 10.20.16.76.17500 > 255.255.255.255.17500: UDP, length 247 4 packets captured 6 packets received by filter 0 packets dropped by kernel **Third party app:** tcpdump
network_traffic_ statistics	Count of the network traffic statistics for specified application by given package or uid	**device_serial:** ADB serial number to identify the device **package_name:** Package name of the application. **uid:** UID (User ID) of the application. Either **package_name** or **uid** must be specified	retcode: tcp_rcv: 2092510 tcp_rcv_pkt: N/Atcp_snd: 170191 tcp_snd_pkt: N/Audp_rcv: N/A udp_rcv_pkt: N/A udp_snd: N/Audp_snd_pkt: N/A
switch_on_mobile_data	Activate the mobile data function. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	empty

Command	Description	Arguments	Return Value
switch_off_mobile_data	Deactivate the mobile data function. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	empty
query_airplane_mode	Get the status of AIRPLANE_MODE	**device_serial:** ADB serial number to identify the device	0 or 1
switch_on_airplane_ mode	Enable airplane mode	**device_serial:** ADB serial number to identify the device	TRUE or FALSE
switch_off_airplane_ mode	Disable airplane mode	**device_serial:** ADB serial number to identify the device	TRUE or FALSE
switch_airplane_mode	Turn on the Airplane Mode if the status is "OFF". Turn off the Airplane Mode if the status is "ON".	**device_serial:** ADB serial number to identify the device **state:** enable / disable	TRUE or FALSE
switch_airplane_ mode_s4	Turn on the Airplane Mode if the status is "OFF". Turn off the Airplane Mode if the status is "ON". (Samsung S4 only)	**device_serial:** ADB serial number to identify the device **state:** enable / disable	TRUE or FALSE

Command	Description	Arguments	Return Value
get_current_apn_ settings	Get current APN settings (APN Id). **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	Returns empty if there's no APN set. Returns the current apn: Row: 0 _id=2878, name=Internet-Beeline, numeric=45207, mcc=452, mnc=07, apn=internet, user=NULL, server=NULL, password=NULL, proxy=NULL, po rt=NULL, mmsproxy=NULL, mmsport=NULL, mmsprotocol=NULL, mmsc=NULL, authtype=-1, type=default,hipri,dun,supl, insert_by=internal, operator=N ULL, area=NULL, state=NULL, identifier=NULL, sim_type=NULL, gid1=NULL, current=1, protocol=IP, roaming_protocol=IP, carrier_enabled=1, bear er=0, mvno_type=, mvno_match_data=, sub_id=-1, profile_id=0, modem_cognitive=0, max_conns=0, wait_time=0, max_conns_time=0, mtu=NULL, databearer=NULL, pppnumber=NULL, dns1=NULL, dns2=NULL, identifier_rule=NULL, pdn_label=NULL, htc_bearer=NULL, htc_tag=NULL

Command	Description	Arguments	Return Value
add_new_apn	Add a new APN record. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device name: APN name, for example T-Mobile US numeric value: The newly added APN should have the same value as the one returned value "getprop gsm.sim.operator.numeric". Otherwise it will not show on the setting UI. apn: Access Point Name (APN) other_fields: Other database fields used in command "contentinsert ". The syntax is: "--bind fieldName:dataType:value". For e.g.:--bind demo:i:123 --bind --demo1:s:'abc'	Returns the newly added APN row: Row: 0 _id=3580, name=VN Mobifone Email, numeric=45207, mcc=452, mnc=07, apn=m-wap, user=, server=, password=, proxy=, port=, mmsproxy=, mmsport=, mmsprotocol=, mmsc=NULL, authtype=-1, type=NULL, insert_by=NULL, operator=NULL, area=NULL, state=NULL, identifier=NULL, sim_type=NULL, gid1=NULL, current=1, protocol=IP, roaming_protocol=IP, carrier_enabled=1, bearer=0, mvno_type=, mvno_match_data=, sub_id=-1, profile_id=0, modem_cognitive=0,max_conns=0, wait_time=0, max_conns_time=0, mtu=NULL, databearer=, pppnumber=, dns1=NULL, dns2=NUL , identifier_rule=NULL, pdn_label=NULL, htc_bearer=NULL, htc_tag=NULL
switch_current_apn	Switch the current APN to specified APN. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device to_apn_id: The id of APN we would like to switch to.	Info: The APN ID "to_apn_id" has successfully switched.
delete_apn	Delete APN by APN ID **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device apn_id: The ID of APN will be deleted.	Info: The APN ID "apn_id" has successfully deleted.
get_max_apn_id	Get the maximum APN Id from database. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	max_apn_id: max_apn_id **Third party app:** sqlite3
sqlite3_installation	Install Sqlite3 on handset to support android database access. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device sqlite3_tool_dir: folder contains sqlite3	emtpy. **Third party app:** sqlite3
restart_handset	Restart specified handsets.	**device_serial:** ADB serial number to identify the device	Restart handset successful.
power_off_handset	Power off specified handsets	**device_serial:** ADB serial number to identify the device	"Power off handset successful" or errors messages.
start_call	Call a designated telephone number	**device_serial:** ADB serial number to identify the device number_to_reach: Phone number to dial	Start call successful.

Command	Description	Arguments	Return Value
answer_call	Answer the CS call	**device_serial:** ADB serial number to identify the device incoming_number: Phone number for the person that is calling	Answer call successful.
end_call	End a CS call	**device_serial:** ADB serial number to identify the device	End call successful.
get_call_status	Check the current voice call status: ringing, conversation terminated, or in conversation	**device_serial:** ADB serial number to identify the device	callState: CALL_STATE_OFFHOOK callState: CALL_STATE_UNKNOWN callState: CALL_STATE_IDLE callState: CALL_STATE_RINGING
send_sms	Send a SMS to a designated mobile phone. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device phone_number: Phone number. message_body: message body	Message has been sent.(success) There were errors during sent message. Please try again later. **Third party app:** chompSMS7.08.apk
get_max_sms_id	Get the maximum SMS Id from database. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	max_sms_id: 1
get_receive_sms_body	Get the body content of received SMS by given contact. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device from_contact	Message body
delete_sms	Delete SMS from specified contact. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device from_contact	SMS have been deleted successfully. **Third party app:** chompSMS7.08.apk
browse_internet	Visit website via default browser	**device_serial:** ADB serial number to identify the device url: URL of the website	Starting: Intent { act=android.intent.action.VIEW dat=http: }
screen_capture	Print the current screen of handset and output to localfolder	**device_serial:** ADB serial number to identify the device screencap_options: Additional screencap options file_name: The file name to save the screenshot.	TRUE or FALSE
start_application	Start a mobile application on Handset	**device_serial:** ADB serial number to identify the device **package_name:** Package name of the application	Events injected: 1 ## Network stats: elapsed time=65ms (0ms mobile, 0ms wifi, 65ms not connected)

Command	Description	Arguments	Return Value
get_element_from_list	Get the specified element from a list	list_data: splitChars: The character(s) will be used to split the list_data by_index: The index of the element will be returned. Default value is "-1", will return all elements.	Return the item in provided list. The output depends on splitChars or by_index: for e.g: A,B,C splitChars=, and by_index=-1 output would be A B C splitChars=, and by_index=1 output would be B splitChars=EMPTY or any character which does not existed in list_data and by_index=-1 output would be A B C
get_imsi_by_handset_id	Get the IMSI responding to the handset Id	**device_serial:** ADB serial number to identify the device	IMSI: 452070...
restart_adb_server	Restart ADB server on the host		* daemon not running. starting it now on port 5037 * * daemon started successfully *
get_4g_3g_id	Get one 4G and one 3G handset Id from the list connecting to the host		3G_handset_id: FA4BBY8000284G _handset_id:FA4BBY985588
get_4g_4g_id	Get two 4G handset Id from the list connecting to the host		4G_handset_id1: FA4BBY8000284 G_handset_id2:FA4BBY985588
get_imei	Get the IMEI of the handset	**device_serial:** ADB serial number to identify the device	IMEI: Phone Subscriber Info: Phone Type = GSM Device ID = *******60208346
execute_adb_command	Execute ADB command	**device_serial:** ADB serial number to identify the device adb_cmd: adb commands to execute	Output depends on provided adb_cmd
execute_sqlite3_command	Execute SQLite3 command. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device sqlite3_command: sqlite3 command	Output depends on provided sqlite3_command **Third party app:** sqlite3
show_apns	Show existing APNs. **Note** Mandatory to be Rooted.	**device_serial:** ADB serial number to identify the device	Shows list of APNs **Third party app:** sqlite3
get_device_ids	Get all device IDs connecting to the host		deviceIds: FA4BBY800028 ff5eybdc

Session profile property settings for ADB custom session

The ADB Custom session is built up by using SSH library, for more details on configuration of this session, see Chapter 81, “SSH Sessions” on page 1522.

CHAPTER 63

OpenStack Neutron Session

OpenStack Neutron session overview

The OpenStack Neutron session allows you to manage OpenStack Neutron Network such as router, network, subnet, port, security group, virtual network topologies including services such as firewalls, load balancers, and virtual private networks (VPNs), via RESTfull HTTP service (see "Working in the REST session window" on page 1180). It supports both interactive mode and executions mode, and may build OpenStack Neutron sessions as follows.

- Build OpenStack Neutron session type using Session Builder, based on REST session quickcall library.

  See Chapter 34, "Session Builder" on page 759 (also video to understand how to use Session Builder) and Chapter 9, "QuickCalls: Defining and using a library of custom actions" on page 205.

- Configure virtual networks based on data retrieved from other session types, topologies or parameter files.
- Use OpenStack Neutron session type to test OpenStack enabled devices.

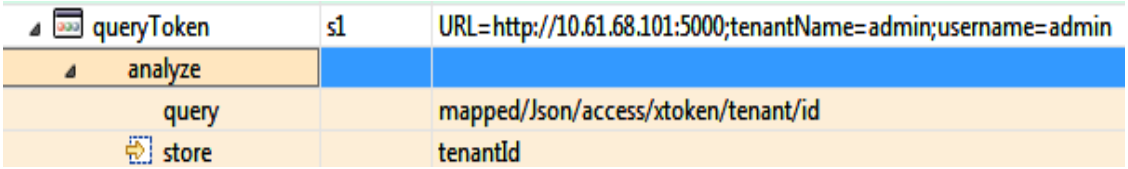
Note The OpenStack Neutron session type supports for OpenStack Network API v2.0.

Authentication and Authorization

The Networking API v2.0 uses the Keystone Identity Service as the default authentication service. When Keystone service is enabled, users MUST provide the authentication token to submit the request to **OpenStack Neutron** service.

With **OpenStack Neutron** session, follow these steps.

- When starting it in Capture mode the QueryToken command should be summited before executing other commands.
- When defining a testcase, insert the QueryToken command right after open step in test case, add an analysis rule to store the **token->tenant_id** into variable so that it can be used in the next steps. For example:



Note When Keystone is enabled, the **tenant_id** attribute is not required in the create command.

Compare REST API JSON with iTest command

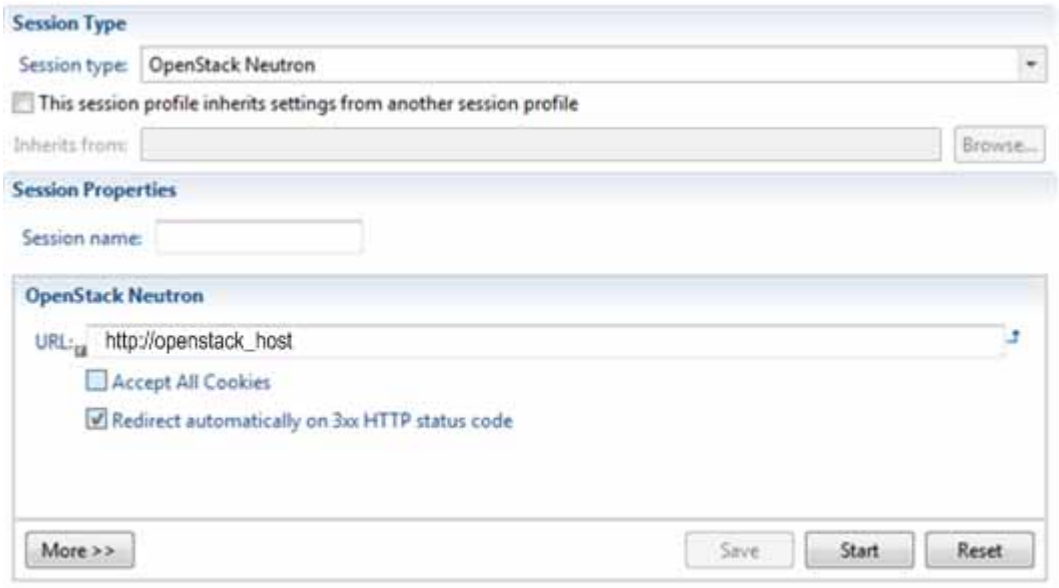
With OpenStack Neutron session, you do not have to know about the complication of JSON request, they're converted to command properties. For example:

The JSON request to create a network:

```
{
    "network": {
        "name": "test_network",
        "admin_state_up": true
    }
}
```

In iTest, start a session in capture mode:

- Provide OpenStack URL in OpenStack Neutron session.
- Click the Start button. As the OpenStack Neutron session is based on Rest session, the GUI is the same as the Rest session GUI

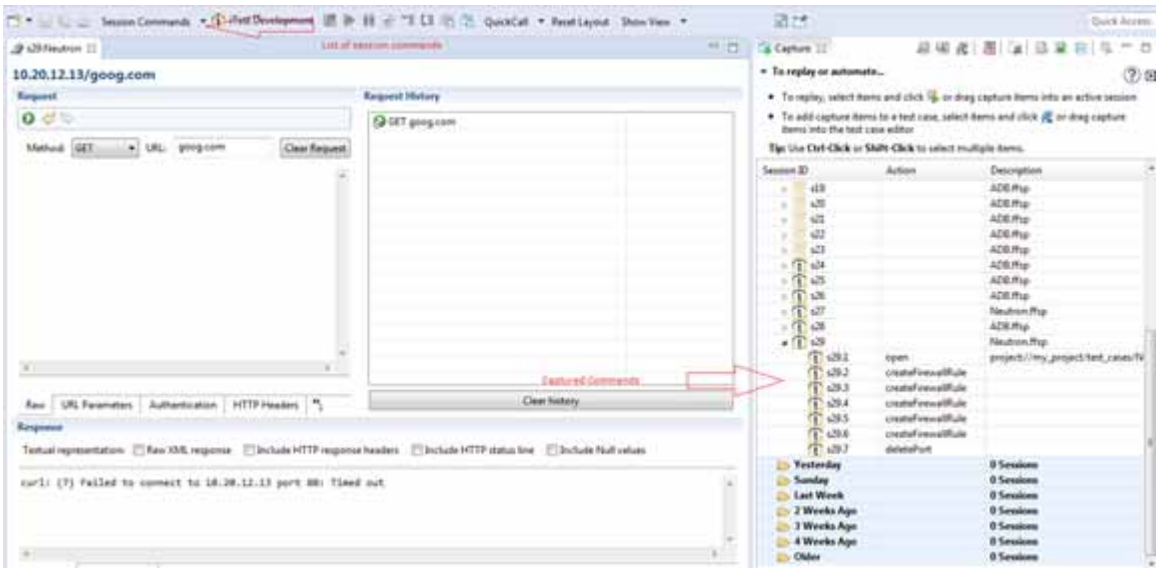


Openstack command set

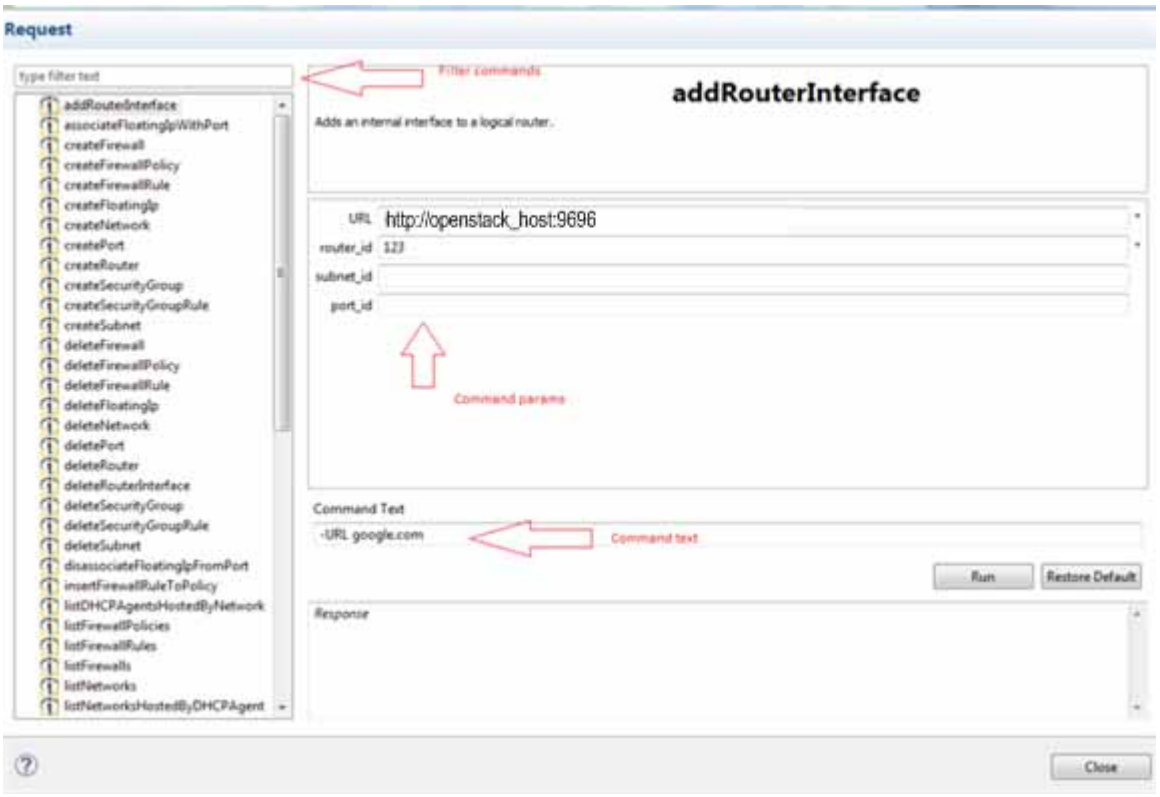
This topic describes most of OpenStack Neutron commands that you can submit from iTest.

While working on an interactive mode in the Session Editor

For OpenStack Neutron session, a Session Command button appears (top of the window), click the button and select a command.



Set command properties



OpenStack command set

Command Name	Description	Arguments
queryToken	Authenticates and generates a token. Must run this command at the beginning.	**URL**: The identity service URL. **tenantName**: The tenant name. username password
listSecurityGroups	Lists all OpenStack Networking security groups to which the specified tenant has access.	**URL**: The neutron service URL.
showSecurityGroup	Shows information for a specified security group.	**URL**: The neutron service URL. **security_group_id**: The unique identifier of the security group.
createSecurityGroup	Create a specified security group.	**URL**: The neutron service URL. **name**: A symbolic name for the security group. Not required to be unique **description:** Describes the security group.
deleteSecurityGroup	The unique identifier of the security group.	**URL**: The neutron service URL. **security_group_id:** The unique identifier of the security group.
listSecurityGroupRules	Lists a summary of all OpenStack Neutron securitygroup rules that the specified tenant can access.	**URL**: The neutron service URL.
showSecurityGroupRule	Shows detailed information for a specified security group rule.	**URL**: The neutron service URL. **rules_security_groups_id:** The unique identifier of the security group rule.

Command Name	Description	Arguments
createSecurityGroup Rule	Creates an OpenStack Neutron security group rule.	**URL**: The neutron service URL. **direction:** Ingress or egress: The direction in which the security group rule is applied. For a compute instance,an ingress security group rule is applied to incoming (ingress) traffic for that instance. An egress rule is applied to traffic leaving the instance. **ethertype:** Must be IPv4 or IPv6, and addresses represented in CIDR must match the ingress or egress rules. **security_group_id:** The security group ID to associate with this security group rule. **port_range_min:** the protocol is TCP or UDP, this value must be less than or equal to the value of the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP type. **port_range_max:** The maximum port number in the range that is matched by the security group rule. The port_range_min attribute constrains the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP type. **protocol:** The protocol that is matched by the security group rule. Valid values are null, tcp, udp, and icmp. **remote_group_id:** The remote group ID to be associated with this security group rule. You can specify either remote_group_id or remote_ip_prefix in the request body. **remote_ip_prefix:** The remote IP prefix to be associated with this security group rule. You can specify either remote_group_id or remote_ip_prefix in the request body. This attribute matches the specified IP prefix as the source IP address of the IP packet.
deleteSecurityGroup Rule	Deletes a specified rule from a OpenStack Networking security group.	**URL**: The neutron service URL. **rules_security_groups_id:** The unique identifier of the security group rule.
listNetworks	Lists networks that the tenant who submits the request can access.	**URL**: The neutron service URL.
showNetwork	Shows information for a specified network.	**URL**: The neutron service URL. **network_id:** The UUID for the network of interest to you.

Command Name	Description	Arguments
createNetwork	Creates a network.	**URL**: The neutron service URL. **name:** The network name. **admin_state_up:** The administrative state of the network , which is up (true) or down (false). **shared:** Indicates whether this network is shared across all tenants. By default, only administrative users can change this value. router_external **provider_physical_network:** The name of the physical network over which the virtual network is implemented for flat and VLAN networks. **provider_network_type:** The physical mechanism by which the virtual network is implemented. Possible values are flat, vlan, local, and gre, corresponding to flat networks, VLAN networks, local networks, and GRE networks as defined above. **provider_segmentation_id:** For VLAN networks, the VLAN VID on the physical network that realizes the virtual network. Valid VLAN VIDs are 1 through 4094. For GRE networks, the tunnel ID. Valid tunnel IDs are any 32 bit unsigned integer. Not applicable to the flat or local network types. **tenant_id:** Admin-only. The UUID of the tenant that will own the network. This tenant can be different from the tenant that makes the create network request.
updateNetwork	Updates a network	**URL**: The neutron service URL. **network_id:** The UUID for the network of interest to you. **name:** The network name. **admin_state_up:** The administrative state of the network , which is up (true) or down (false). **shared:** Admin-only. Indicates whether this network is shared across all tenants.
deleteNetwork	Deletes a specified network and its associated resources.	**URL**: The neutron service URL. **network_id:** The UUID for the network of interest to you.
listSubnets	Lists all subnets that are accessible to the tenant who submits the request.	**URL**: The neutron service URL.
showSubnet	Gets information about a specified subnet.	**URL**: The neutron service URL. **subnet_id:** The UUID for the subnet of interest to you.

Command Name	Description	Arguments
createSubnet	Creates a subnet on the specified network.	**URL**: The neutron service URL. **cidr:** The CIDR. **name:** The subnet name. **ip_version:** The IP version, which is 4 or 6. **allocation_pools_start:** The start addresses for the allocation pools. **allocation_pools_end:** The end addresses for the allocation pools. **gateway_ip:** The gateway IP address. **enable_dhcp:** Set to true if DHCP is enabled and false if DHCP is disabled. **network_id:** The ID of the attached network. **tenant_id:** The ID of the tenant who owns the network. Only administrative users can specify a tenant ID other than their own. **id:** The ID of the subnet.
updateSubnet	Updates the specified subnet.	**URL**: The neutron service URL. **subnet_id:** The UUID for the subnet of interest to you. **name:** The subnet name. **gateway_ip:** The gateway IP address. **enable_dhcp:** Set to true if DHCP is enabled and false if DHCP is disabled.
deleteSubnet	deleteSubnet	**URL**: The neutron service URL. **subnet_id:** The UUID for the subnet of interest to you.
listPorts	Lists ports to which the tenant has access.	**URL**: The neutron service URL.
showPort	Shows information for a specified port.	**URL**: The neutron service URL. **port_id:** The UUID for the port of interest to you.

Command Name	Description	Arguments
createPort	Creates a port on a specified network.	**URL**: The neutron service URL. **name:** A symbolic name for the port. **admin_state_up:** The administrative status of the port, which is up (true) or down (false). **fixed_ip_address:** If you specify only a subnet ID, OpenStack Networking allocates an available IP from that subnet to the port. If you specify both a subnet ID and an IP address, OpenStack Networking tries to allocate the specified address to the port. **fixed_subnet_id:** The fixed subnet id. **mac_address:** The MAC address. If you do not specify a MAC address, OpenStack Networking tries to allocate one. **security_groups:** Security groups. Specify one or more security group IDs. **device_id:** The ID of the device that uses this port. For example, a virtual server. **device_owner:** The ID of the entity that uses this port.For example, a dhcp agent. **network_id:** The ID of the network. **tenant_id:** The ID of the tenant who owns the network. Only administrative users can specify a tenant ID other than their own. **id:** The ID of the port.
updatePort	Updates a specified port.	**URL**: The neutron service URL. **port_id:** The UUID for the port of interest to you. **name:** A symbolic name for the port. **admin_state_up:** The administrative status of the port, which is up (true) or down (false). **fixed_ip_address:** Fixed IP address **fixed_subnet_id:** Fixed subnet id. **security_groups:** Security groups. Specify one or more security group IDs. **device_id:** The ID of the device that uses this port. For example, a virtual server. **device_owner:** The ID of the entity that uses this port. For example, a dhcp agent.
deletePort	Removes a specified port from an OpenStack Neutron network.	**URL**: The neutron service URL. **port_id:** The UUID for the port of interest to you.

Command Name	Description	Arguments
createRouter	Creates a logical router	**URL**: The neutron service URL. **name:** The router name. **admin_state_up:** The administrative state of the router, which is up (true) or down (false). **network_id:** The network ID. **tenant_id:** The tenant ID.
updateRouter	Updates a logical router.	**URL**: The neutron service URL. **router_id:** The UUID of the router. **name:** The router name. **admin_state_up:** The administrative state of the router, which is up (true) or down (false). **network_id:** The network ID.
deleteRouter	Deletes a logical router and, if present, its external gateway interface.	**URL**: The neutron service URL. **router_id:** The UUID of the router.
showRouter	Shows details for a specified router.	**URL**: The neutron service URL. **router_id:** The UUID of the router.
listRouters	List routers that belong to a given tenant.	**URL**: The neutron service URL.
addRouterInterface	Adds an internal interface to a logical router.	**URL**: The neutron service URL. **router_id:** The UUID of the router. **subnet_id:** The subnet ID. **port_id:** The UUID for the port.
deleteRouterInterface	Removes an internal interface from a logical router.	**URL**: The neutron service URL. **router_id:** The UUID of the router. **subnet_id:** The subnet ID. **port_id:** The UUID for the port.
createFloatingIp	Creates a floating IP, and, if you specify port information, associates the floating IP with an internal port.	**URL**: The neutron service URL. **floating_network_id:** The ID of the network associated with the floating IP. **fixed_ip_address:** The fixed IP address associated with the floating IP. **port_id:** The port ID. **tenant_id:** The tenant ID.
associateFloatingIp WithPort	Updates a floating IP and its association with an internal port.	**URL**: The neutron service URL. **floatingip_id:** The UUID of the floating IP. **port_id:** The port ID.
disassociateFloatingI pFromPort	To disassociate a floating IP from an internal port	**URL**: The neutron service URL. **floatingip_id:** The UUID of the floating IP.

Command Name	Description	Arguments
deleteFloatingIp	Deletes a floating IP and, if present, its associated port.	**URL**: The neutron service URL. **floatingip_id:** The UUID of the floating IP.
showFloatingIp	Shows details for a specified floating IP.	**URL**: The neutron service URL. floatingip_id The UUID of the floating IP.
createFirewall	Creates a firewall.	**URL**: The neutron service URL. **firewall_policy_id:** The firewall policy uuid that this firewall is associated with. **name:** Human readable name for the firewall (255 characters limit). **description:** Human readable description for the firewall (1024 characters limit). **shared:** When set to True makes this firewall rule visible to tenants other than its owner, and can be used in firewall policies not owned by its tenant. **admin_state_up:** The administrative state of the network, which is up (true) or down (false). **tenant_id:** The tenant ID.
deleteFirewall	Deletes a firewall.	**URL**: The neutron service URL. **firewall_id:** Unique identifier for the firewall object.
updateFirewall	Updates a firewall, provided status is not PENDING_*.	**URL**: The neutron service URL. **firewall_id:** Unique identifier for the firewall object. **firewall_policy_id:** The firewall policy uuid that this firewall is associated with. **name:** Human readable name for the firewall (255 characters limit). **description:** Human readable description for the firewall (1024 characters limit). **admin_state_up:** The administrative state of the network, which is up (true) or down (false). **shared:** When set to True makes this firewall rule visible to tenants other than its owner, and can be used in firewall policies not owned by its tenant.
listFirewalls	Lists firewalls.	**URL**: The neutron service URL.
showFirewall	Shows firewall details.	**URL**: The neutron service URL. **firewall_id:** Unique identifier for the firewall object.

Command Name	Description	Arguments
createFirewallRule	Creates a firewall rule.	**URL**: The neutron service URL. **action_rule:** Action to be performed on the traffic matching the rule. (allow, deny) **destination_ip_address:** Destination IP address or CIDR. **destination_port:** Destination port number or a range. enabled When set to False will disable this rule in the firewall policy. **ip_version:** IP Protocol Version. **name:** Human readable name for the firewall rule (255 characters limit). **description:** Human readable description for the firewall Rule (1024 characters limit). **protocol:** IP Protocol. **shared:** When set to True makes this firewall rule visible to tenants other than its owner, and can be used in firewall policies not owned by its tenant. **source_ip_address:** Source IP address or CIDR. **source_port:** Source port number or a range. **tenant_id:** The tenant ID.
updateFirewallRule	Updates a firewall rule.	**URL**: The neutron service URL. **firewall_rule_id:** Unique identifier for the firewall rule object. **action_rule:** Action to be performed on the traffic matching the rule (allow, deny) **destination_ip_address:** Destination IP address or CIDR. **destination_port:** Destination port number or a range. **enabled:** When set to False will disable this rule in the firewall policy. **ip_version:** IP Protocol Version. **name:** Human readable name for the firewall rule (255 characters limit). **description:** Human readable description for the firewall Rule (1024 characters limit). **protocol:** IP Protocol. **shared:** When set to True makes this firewall rule visible to tenants other than its owner, and can be used in firewall policies not owned by its tenant. **source_ip_address:** Source IP address or CIDR. **source_port:** Source port number or a range.

Command Name	Description	Arguments
deleteFirewallRule	Deletes a firewall rule.	**URL**: The neutron service URL. **firewall_rule_id:** Unique identifier for the firewall rule object.
listFirewallRules	Lists firewall rules.	**URL**: The neutron service URL.
showFirewallRule	Shows firewall rule details.	**URL**: The neutron service URL. **firewall_rule_id:** Unique identifier for the firewall rule object.
createFirewallPolicy	Creates a firewall policy.	**URL**: The neutron service URL. **name:** Human readable name for the firewall policy (255 characters limit). **audited:** When set to True by the policy owner indicates that the firewall policy has been audited. **description:** Human readable description for the firewall policy (1024 characters limit) **firewall_rule_id:** Unique identifier for the firewall rule object. **shared:** When set to True makes this firewall policy visible to tenants other than its owner. **tenant_id:** Owner of the firewall policy. Only admin users can specify a tenant identifier other than their own.
updateFirewallPolicy	Updates a firewall policy.	**URL**: The neutron service URL. **firewall_policy_id:** Unique identifier for the firewall policy object. **name:** Human readable name for the firewall policy (255 characters limit). **audited:** When set to True by the policy owner indicates that the firewall policy has been audited. **description:** Human readable description for the firewall policy (1024 characters limit) **firewall_rule_id:** Unique identifier for the firewall rule object. **shared:** When set to True makes this firewall policy visible to tenants other than its owner.
deleteFirewallPolicy	Deletes a firewall policy.	**URL**: The neutron service URL. **firewall_policy_id:** Unique identifier for the firewall policy object.
listFirewallPolicies	Lists firewall policies.	**URL**: The neutron service URL.
showFirewallPolicy	Shows firewall policy details.	**URL**: The neutron service URL. **firewall_policy_id:** Unique identifier for the firewall policy object.

Command Name	Description	Arguments
insertFirewallRuleToPolicy	Inserts a firewall rule in a firewall policy relative to the position of other rules.	**URL**: The neutron service URL. **firewall_policy_id:** Unique identifier for the firewall policy object. **firewall_rule_id:** Unique identifier for the firewall rule object. **insert_after:** insert_after parameter refer to firewall rule uuids already associated with the firewall policy. **insert_before:** insert_before parameter refer to firewall rule uuids already associated with the firewall policy.
removeFirewallRuleFromPolicy	Removes a firewall rule from a firewall policy.	**URL**: The neutron service URL. **firewall_policy_id:** Unique identifier for the firewall policy object. **firewall_rule_id:** Unique identifier for the firewall rule object.
listNetworksHostedByDHCPAgent	Lists networks that the specified DHCP agenthosts.	**URL**: The neutron service URL. **agent_id:** The agent ID.
listDHCPAgentsHostedByNetwork	Lists DHCP agents that hosts a specified network.	**URL**: The neutron service URL. **network_id:** The network ID.
scheduleNetworkToDHCPAgent	Schedules the network to that the specified DHCP agent.	**URL**: The neutron service URL. **agent_id:** The agent ID. **network_id:** The network ID.
removeNetworkFromDHCPAgent	Removes the network from that the specified DHCP agent.	**URL**: The neutron service URL. **agent_id:** The agent ID. **network_id:** The network ID.

Send command properties

The command properties which were modified or their default value are non-empty will be translated to JSON request and sent to OpenStack Neutron Service.

For example: With **createNetwork** above, only **URI**, **name**, **tenant_id**, and **admin_state_up** are sent to OpenStack Neutron Service.

Response format

OpenStack Neutron session returns the response in JSON format, exactly as what are returned from OpenStack Neutron Service.

Tip Any command that you define in the OpenStack Neutron session test case is forwarded to the specified OpenStack Neutron Service.

Session profile property settings for OpenStack session

The OpenStack session is built up by using REST QuickCall library, for more details on configuration of this session, see "Session profile property settings for REST sessions" on page 1191.

CHAPTER 63

CloudStress Session

CloudStress Session Overview

The iTest CloudStress session provides way to automate testing and performance verification of your cloud infrastructure, without having to know scripting languages.

The steps below describes a quick start process to operate the iTest CloudStress session. See "CloudStress Session" on page 1249 and "CloudStress Session Command Set" on page 1253.

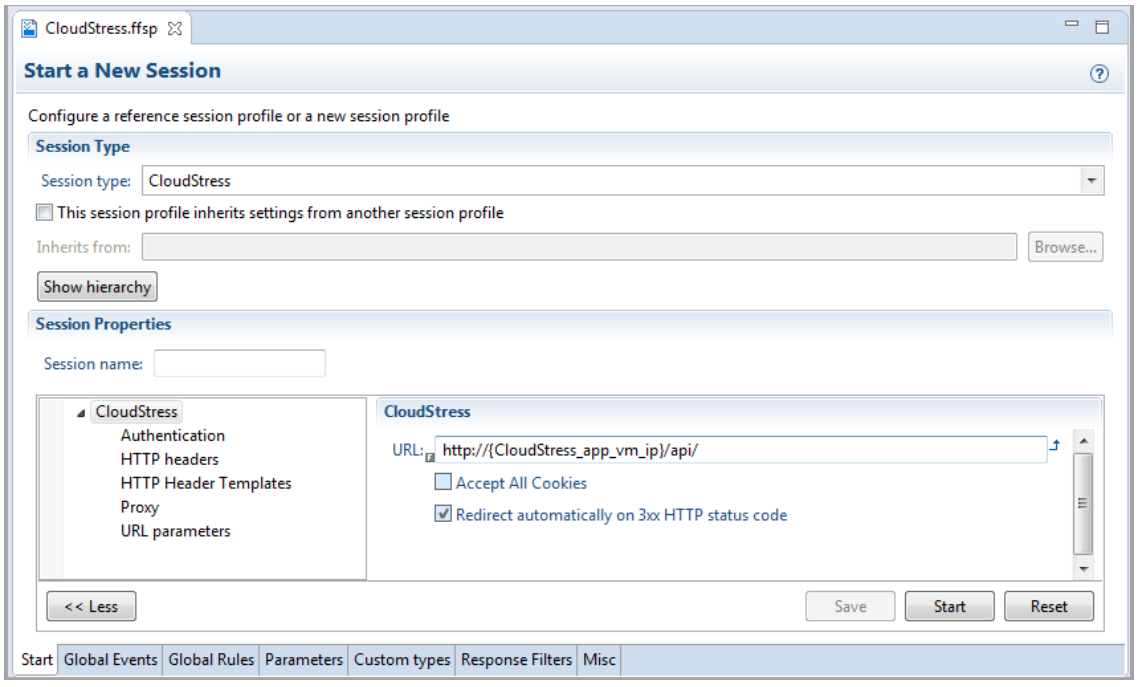
CloudStress Session

In iTest, open a session, select CloudStress, and specify the required parameter values:

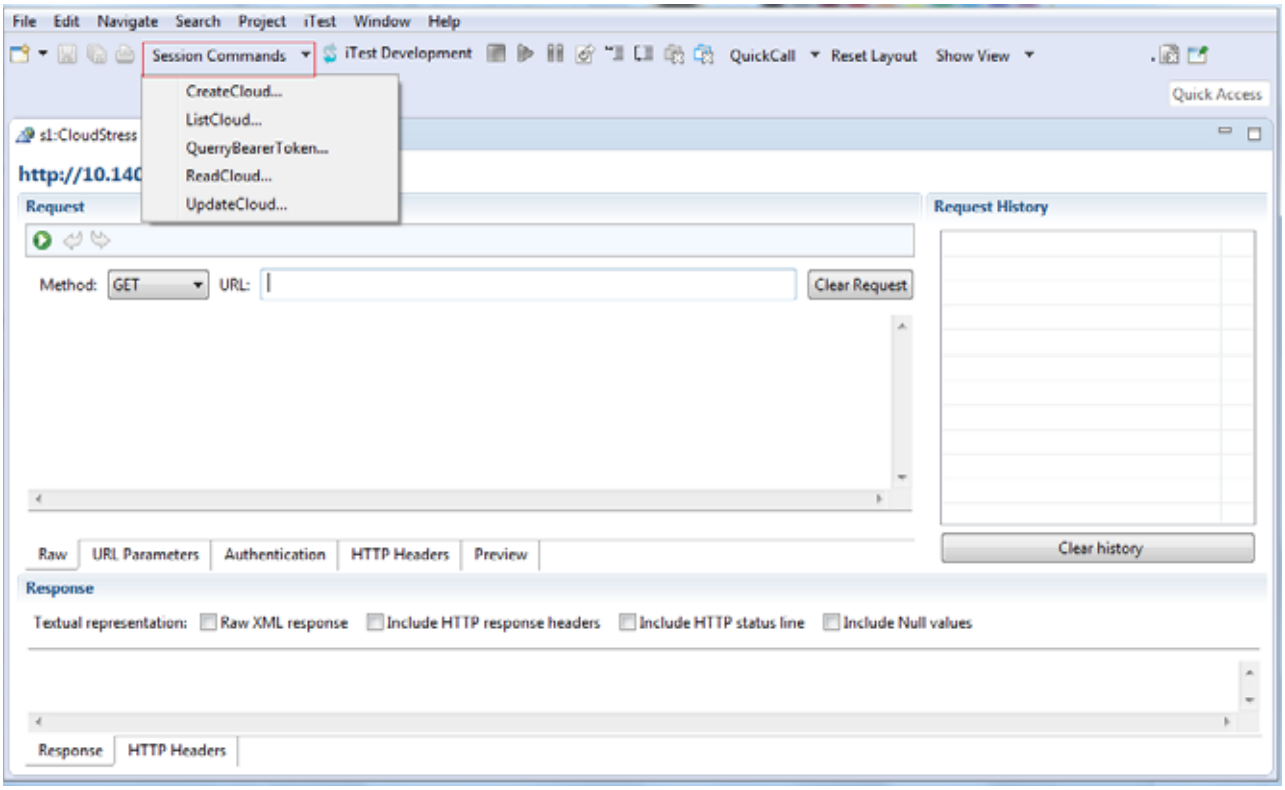
- **URL:** The CloudStress application VM URL, for example: http://cloudstress-app-1.spirent.com/api.
- **Authentication:** Leave blank—Temeva credentials are supplied using QueryBearerToken once the session is open.

All other settings can be left with their defaults, except for **Proxy**, which may be required if reaching Temeva and the CloudStress application via a proxy server.

Note The CloudStress session requires a license.



Click **Start** and the session window displays with session commands.



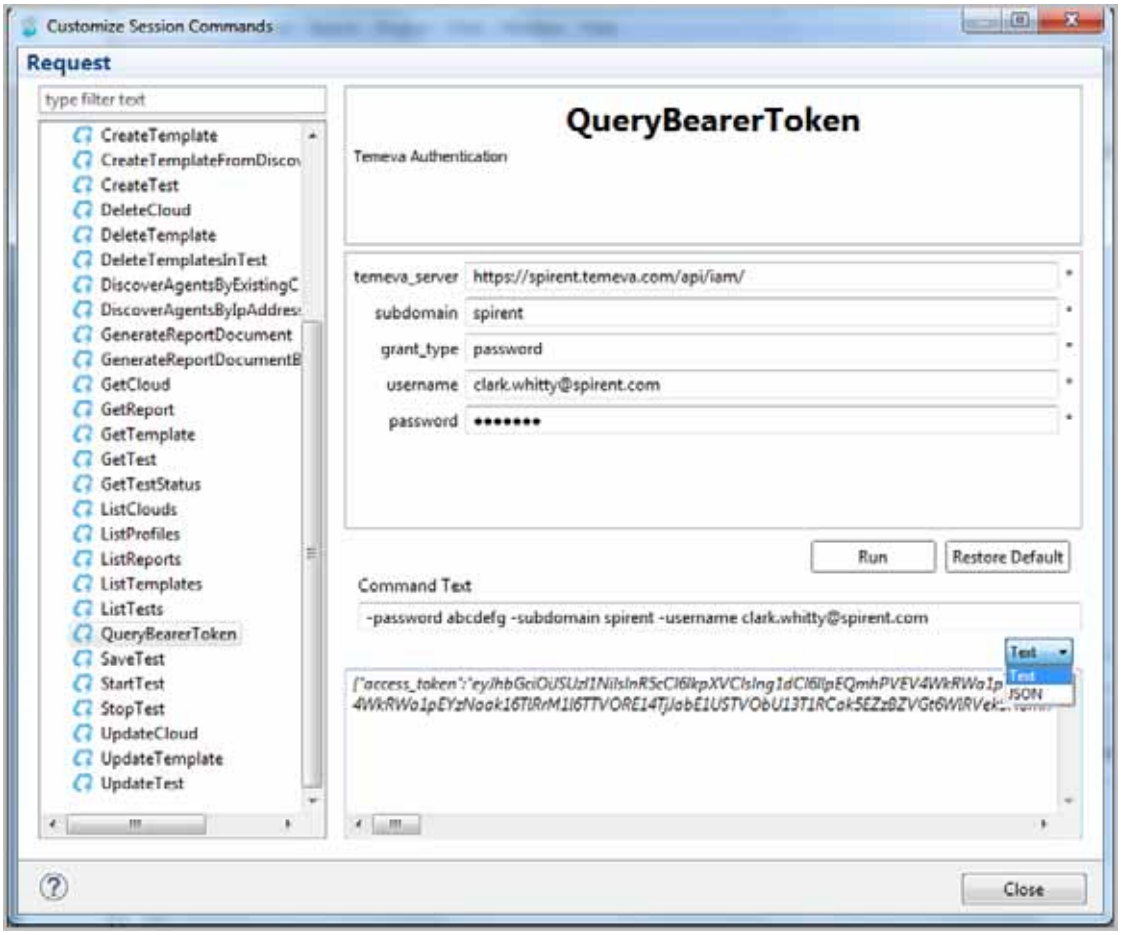
Choose the **QueryBearerToken** session command to authenticate to Temeva. Enter user credentials for the Temava account associated with the test, click **Run**, and verify that an access token is returned in the response window.

On the **Response** tab/section, the default display format is auto-detected as **JSON** or **Text** form.

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.
- If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

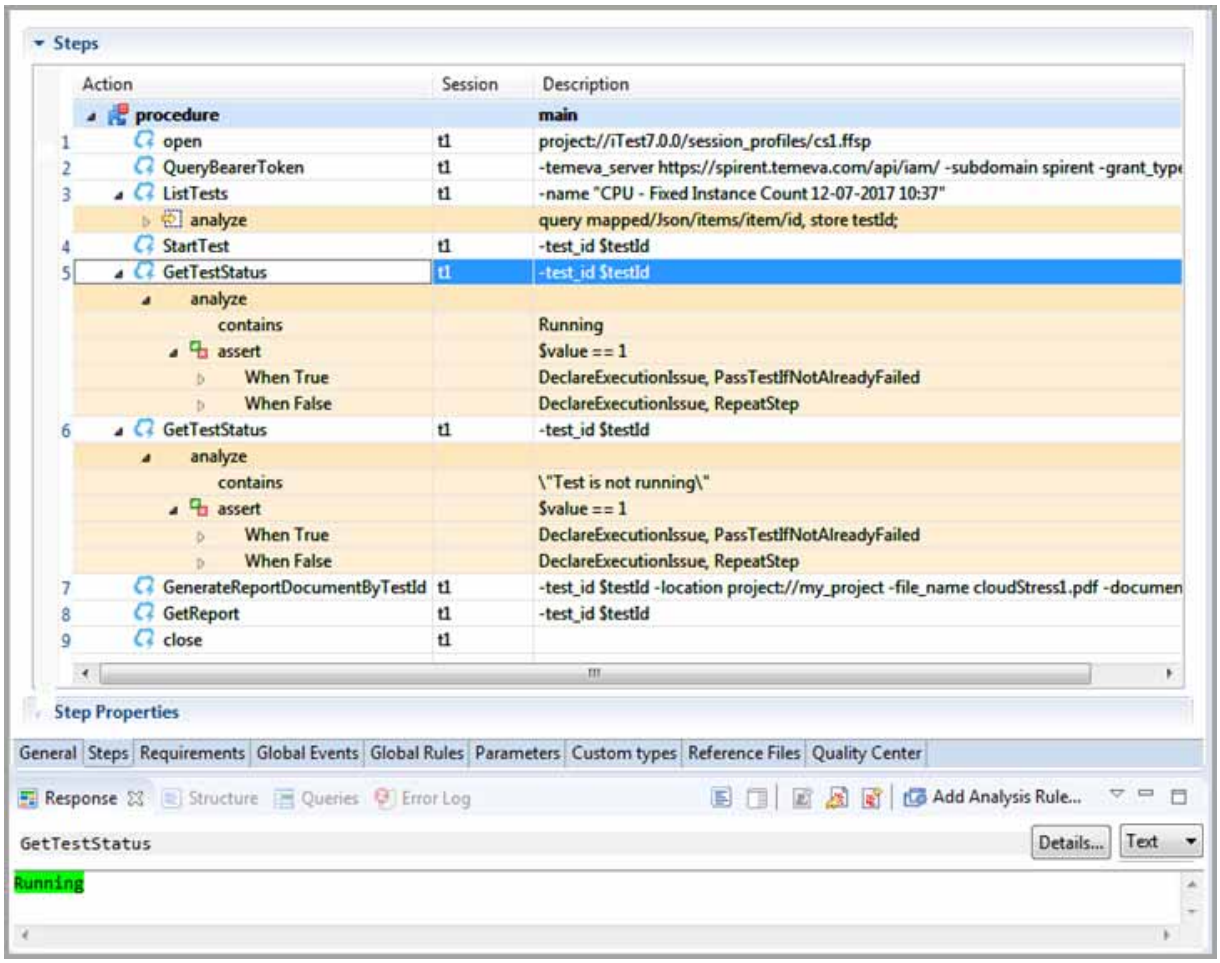
Click **JSON**/**Text** options from the dropdown list on the **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

Note When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 203 in Chapter 8, "JSON Editor").

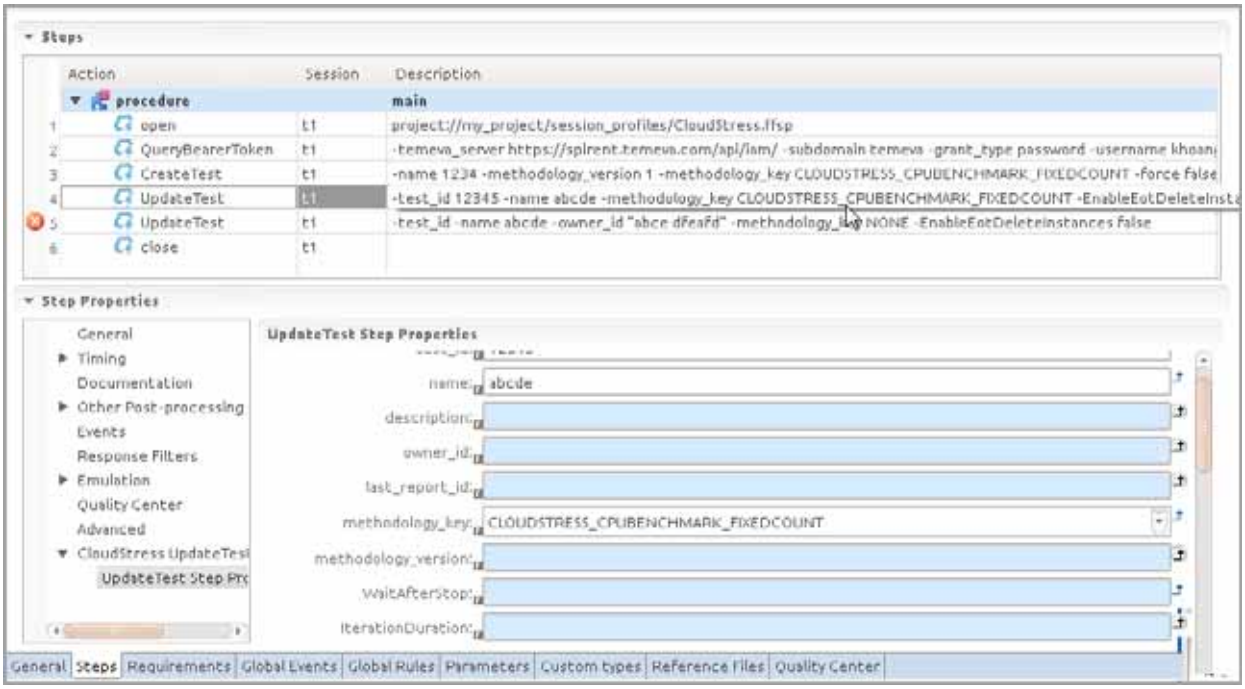


Run a sequence of session commands to accomplish your tasks. For example, to run an existing test, the following sequence would follow:

- **ListTests** (to get the test id of the existing test)
- **StartTest** (invoking the test to run)
- **GetTestStatus** (to verify the test has started and ultimately stopped)
- **GenerateReportDocumentByTestId** (to download a test report in PDF, DOCX, or XLSX format)
- **GetReport** (to access specific report data elements in JSON format)



The input argument value in Step properties mirror the Steps Description column.



CloudStress Session Command Set

This topic lists the CloudStress commands that you can submit from iTest.

The CloudStress sessions provides the following capabilities:

- Authentication to Temeva
- Create, read, update, list, and delete clouds
- Discover CloudStress agent VMs by IP address and port ranges
- Discover CloudStress agent VMs by selecting an existing cloud
- Create, read, update, list, and delete templates
- Create, read, update, and list tests
- Start, query status, and stop a test execution
- Generate and download reports in PDF, DOCX, and XLSX format
- Generate and response-map reports in JSON format

Authentication command (Mandatory)

Commands	Description	Arguments
QueryBearerToken	Must run this API at the beginning Get token for iTest Cloud Stress session	temeva_server: The identity service URL (ex: https://spirent.temeva.com/api/iam/) subdomain: (ex: spirent) grant_type: (ex: password) username: the username that registered on temeva_server password: password of user

Cloud commands

Commands	Description	Arguments
CreateCloud	Creates a new cloud Override an exiting cloud (force = true)	Name: is_credentials_saved Provider: provider_server provider_tenant provider_tenant_domain provider_user provider_password provider_user_domain force
UpdateCloud	Update an existing cloud	cloud_id name is_credentials_saved provider provider_server provider_tenant provider_tenant_domain provider_user provider_password provider_user_domain
ListClouds	Get list of clouds	id name metadata_only
GetCloud	Get a cloud by id	cloud_id
DeleteCloud	Delete a cloud	cloud_id

Discovery commands

Commands	Description	Arguments
DiscoverAgents ByIpAddress	Get Agents by IP Adress	agent_ips agent_ip_counts is_save_credentials ports port_counts template_id
DiscoverAgentsBy ExistingCloud	Get Agents by existing Cloud	cloud_id is_save_credentials provider_user provider_password provider_user_domain

Template commands

Commands	Description	Arguments
CreateTemplate	Creating a Template	template_name owner_id res_id cloud_id agent_ips discovery_id model_name nodes instances is_consistent is_credentials_saved is_shared force
CreateTemplate FromDiscover	Creating a template	discover_id template_name res_id
GetTemplate	Get a template by id	template_id

Commands	Description	Arguments
UpdateTemplate	Update existing template	template_id* template_name owner_id res_id cloud_id agent_ips discovery_id model_name nodes instances is_save_credentials is_consistent is_shared force
ListTemplates	Get list of template	id name metadata_only
DeleteTemplate	Delete a template	template_id

Test commands

Commands	Description	Arguments
CreateTest	Create a test	name description owner_id last_report_id methodology_version methodology_key(CLOUDSTRESS_CPUBENCHMARK_FIXEDCOUNT, CLOUDSTRESS_MEMORYBENCHMARK_FIXEDCOUNT, CLOUDSTRESS_STORAGEBENCHMARK_FIXEDCOUNT, CLOUDSTRESS_NETWORKBENCHMARK_FIXEDCOUNT, CLOUDSTRESS_BENCHMARK_FIXEDCOUNT, CLOUDSTRESS_CPUBENCHMARK_FIXEDLOAD, CLOUDSTRESS_MEMORYBENCHMARK_FIXEDLOAD, CLOUDSTRESS_STORAGEBENCHMARK_FIXEDLOAD, CLOUDSTRESS_NETWORKBENCHMARK_FIXEDLOAD, CLOUDSTRESS_BENCHMARK_FIXEDLOAD) force
UpdateTest	Update test's properties	test_id name description owner_id last_report_id methodology_key methodology_version IterationDuration InitialSearchValue SearchResolution WaitAfterStop ProviderRespMaxWaitTime ProviderRespPollTime DeployMaxWaitTime DeployPollTime TearDownMaxWaitTime TearDownPollTime DeployBatchSize TearDownBatchSize EnableEotDeleteInstances
ConfigureTest Property	Configure a specified property of test.	test_id*: String property*: String property_value: String

Commands	Description	Arguments
ConfigStorage BenchmarkTest Properties	Configure Storage Benchmark Test's Properties	MinMaxStorageLoadParams WaitAfterStop IterationDuration InitialSearchValue MaxSearchValue MinSearchValue SearchResolution AcceptableStorageReadVariance AcceptableStorageReadLatency AcceptableStorageWriteVariance AcceptableStorageWriteLatency StorageReadVarianceSuccessThreshold StorageReadLatencySuccessThreshold StorageWriteVarianceSuccessThreshold StorageWriteLatencySuccessThreshold
ConfigCPU BenchmarkTest Properties	Configure CPU Benchmark Test's Properties	MinMaxCpuLoadParam WaitAfterStop SearchResolution InitialSearchValue MinSearchValue MaxSearchValue IterationDuration AcceptableVariance SuccessThreshold
ConfigNetwork BenchmarkTest Properties	Configure Network Benchmark Test's Properties	MinMaxNetworkLoadParams WaitAfterStop SearchResolution InitialSearchValue MinSearchValue MaxSearchValue IterationDuration AcceptableNetworkReadVariance AcceptableNetworkReadLatency AcceptableNetworkWriteVariance AcceptableNetworkWriteLatency NetworkReadVarianceSuccessThreshold NetworkReadLatencySuccessThreshold NetworkWriteVarianceSuccessThreshold NetworkWriteLatencySuccessThreshold

Commands	Description	Arguments
ConfigMemory BenchmarkTest Properties	Configure Memory Benchmark Test's Properties	MinMaxMemoryLoadParams WaitAfterStop IterationDuration InitialSearchValue MaxSearchValue MinSearchValue SearchResolution AcceptableMemoryReadVariance AcceptableMemoryReadLatency AcceptableMemoryWriteVariance AcceptableMemoryWriteLatency MemoryReadVarianceSuccessThreshold MemoryReadLatencySuccessThreshold MemoryWriteVarianceSuccessThreshold MemoryWriteLatencySuccessThreshold
ConfigBenchmark TestProperties (continued on the next row)	Configure Benchmark Test's Properties	EnableCpuLoad EnableCpuTestCriteria EnableMemoryLoad EnableMemoryTestCriteria EnableNetworkLoad EnableNetworkTestCriteria EnableStorageLoad EnableStorageTestCriteria IterationDuration InitialSearchValue SearchResolution WaitAfterStop ProviderRespMaxWaitTime ProviderRespPollTime DeployMaxWaitTime DeployPollTime TearDownMaxWaitTime TearDownPollTime DeployBatchSize TearDownBatchSize EnableEotDeleteInstances MinMaxCpuLoadParam MinMaxMemoryLoadParams MinMaxNetworkLoadParams MinMaxStorageLoadParams AcceptableCpuLoadVariance CpuLoadVarianceSuccessThreshold

Commands	Description	Arguments
ConfigBenchmark TestProperties (continued from the previous row)		AcceptableMemoryReadVariance AcceptableMemoryReadLatency AcceptableMemoryWriteVariance AcceptableMemoryWriteLatency MemoryReadVarianceSuccessThreshold MemoryReadLatencySuccessThreshold MemoryWriteVarianceSuccessThreshold MemoryWriteLatencySuccessThreshold AcceptableNetworkReadVariance AcceptableNetworkReadLatency AcceptableNetworkWriteVariance AcceptableNetworkWriteLatency NetworkReadVarianceSuccessThreshold NetworkReadLatencySuccessThreshold NetworkWriteVarianceSuccessThreshold NetworkWriteLatencySuccessThreshold AcceptableStorageReadVariance AcceptableStorageReadLatency AcceptableStorageWriteVariance AcceptableStorageWriteLatency StorageReadVarianceSuccessThreshold StorageReadLatencySuccessThreshold StorageWriteVarianceSuccessThreshold StorageWriteLatencySuccessThreshold
GetTest	Get a test by id	test_id
ListTests	Get list of test	id name cursor metadata_only page_size
SaveTest	After the test has been configured, it can be saved by performing a POST To save or update an existing test perform a PUT	test_id Response:
StartTest	Run an existing test	test_id
GetTestStatus	Retrieve information on the execution status of the test	test_id

Commands	Description	Arguments
StopTest	Test can be stopped before completion if needed. This can be accomplished by deleting the execution resource associated with a running test	execution_id
ApplyProfileToTest	Add a specified profile to a specified test	test_id template_id profile_id
ConfigureTemplates InTest	Change machine's properties of test	test_id template_index machine_id count machine_name
AddTemplateToTest	Add specified template to specified test	test_id index template_id count
DeleteTemplatesIn Test	Remove test's machines	test_id template_index

Result commands

Commands	Description	Arguments
GenerateReport Document	Download a document using report id Supported types: PDF, DOCX, XLSX	report_id location file_name document_type
GenerateReport DocumentByTestId	Download a latest document of test using test id Supported types: PDF, DOCX, XLSX	test_id location file_name document_type
ListReports	List test reports of specified test	id name test_id metadata_only
getReport	Get report as JSON document	test_id report_id
ListProfiles	List all profiles	id name owner_id type metadata_only

Session profile property settings for CloudStress session

The CloudStress session is built up by using REST library, for more details on configuration of this session, see Chapter 62, “REST sessions” on page 1180.

CHAPTER 53

Zephyr for JIRA

Overview

Zephyr for JIRA Test Management system with the add-on **ZAPI**, integrated with **iTest** makes it easier for you to share your projects across the organization. The integration (**Zephyr for JIRA** and **iTest**) allows you to work with JIRA tests via RESTful APIs to perform these tasks.

Note In order to understand more about **Zephyr for JIRA** refer to the Zephyr Overview. See also the related documentation.

- Execute test in iTest and upload results to specified **Zephyr for JIRA** test after execution.
- Update the uploaded tests steps in **Zephyr for JIRA**.
- Create Cycles test cycles in **Zephyr for JIRA**

ZAPI add-on to **Zephyr for JIRA**, via RESTful APIs, allows access to Zephyr's testing data including the ability to view and upload iTest execution results data. This can be used to query test cycles, fetch tests, and test cycles, update test results, create new tests, etc.

Note See Zapi documentation for detailed information.

How iTest and Zephyr for JIRA work together

iTest is integrated with **Zephyr for JIRA**. After setting up as described in "Configure iTest and Zephyr for JIRA" on page 1067, you can start test execution and monitor the results in both applications.

- In the **Zephyr for JIRA** Workspace you may create, plan, execute and track tests.

  Create a test (similar to any other issue type in JIRA), which has additional fields and capabilities. Similar to any other JIRA issue type, tests belong to a project.
- You can define parameters and set their values in either **Zephyr for JIRA** or **iTest.**
- You can associate any iTest test case with a Test Case in the **Zephyr for JIRA** test.
- iTest can upload test case step information to the specified **Zephyr for JIRA** test.
- iTest can publish test report data to the specified **Zephyr for JIRA** test cycle. instance.
- Reports in iTESTRT server may be uploaded **Zephyr for JIRA** by proving **--zephyr** option along with the required parameters file.

Configure iTest and Zephyr for JIRA

The iTest integration with **Zephyr for JIRA** requires the following set up prior to working with iTest and **Zephyr for JIRA**.

Note For the supported versions of Zephyr for JIRA and and add-on ZAP see these links:

- https://marketplace.atlassian.com/plugins/com.thed.zephyr.je/versions
- https://marketplace.atlassian.com/plugins/com.thed.zephyr.zapi/versions

---

1. Set up a JIRA server (JIRA cloud or JIRA Server).

   Refer to JIRA cloud/JIRA Server installation instructions.

2. Install **Zephyr for JIRA**.

   Refer to the **Zephyr for JIRA** installation instructions.

3. Install **Zapi** for **Zephyr for JIRA.**

   Refer to the **Zapi** for **Zephyr for JIRA** installation instructions

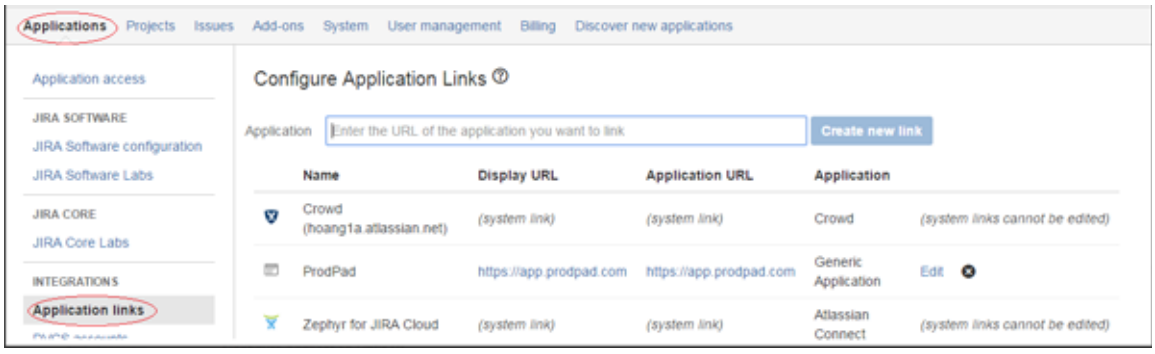
4. Configure iTEST and Zephyr for JIRA Application Links

   a. Create a public/private key pair with **openssl**.

   b. Connect the iTest plugin to JIRA

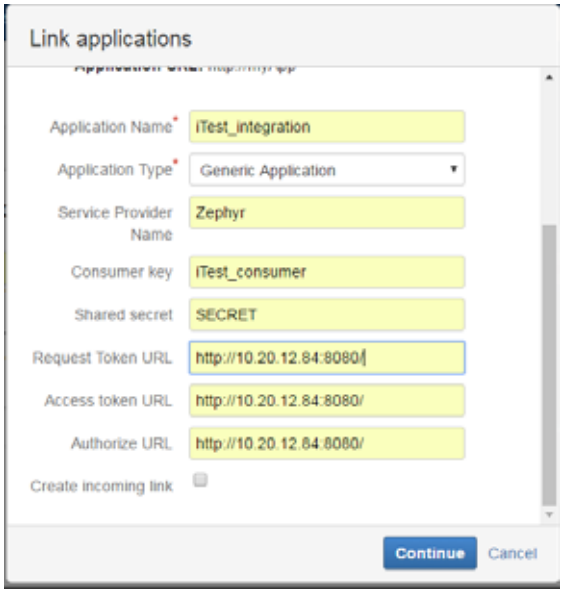
   Select **Add-ons > Application Links** (In **Zephyr for JIRA**) in the administration section. The **Configure Application Links** page displays.

   - Enter the URL of the application you want to link in the text box and clicking create new link (you may also enter **localhost**).



   - Choose an application name (e.g., iTest_integration), consumer key (e.g. iTest_consumer) and use the generated public key from Step a, above.

- Specify the server URL for Request Token URL, Access token URL and authorize URL.



For details on configuring application link refer to the Atlassian document.

c Customize **Zephyr on JIRA** test status.

Customize status for Executions Step as required.

Select **Add-ons > Application Links** (In **Zephyr for JIRA**) in the administration section.

Click **Customize Test Status** or **Customize Step Status** tab on the left and the following screen displays.

Edit existing status and add new status as required.

Provide name = INDETERMINATE, add a description, and pick a color to indicate the status, and click on Add.

Note You cannot delete any existing status, only edit them. You can delete any new status that you add.

5 Configure the parameters file in iTest

All the Zephyr server information must be stored in parameter files. These parameters are used during execution and the required data uploaded to Zephyr on JIRA. The table below list the required parameters and type of value that should be included in the iTest parameter file. See "Create iTest parameter file with Zephyr for JIRA Information" on page 1071.

Parameter Name	Description	Data Type	Example
zephyr_type	**JIRA** for use with **Zephyr for Jira**.	String	jira
zephyr_report_path	HTML report path to upload reports to **JIRA**	String	D:/my_project/test_cases/test_zephyr.html
zephyr_cycle_name	Zephyr Cycle name. Automatically created, if does not exist.	String	myLittleCyle
zephyr_base_url	Base url of the JIRA server.	URL	https://jira.spirenteng.com
jira_version_name	The name of the JIRA project version. Refer to Project Versions.	String	1.0, myVersion
jira_consumer_private_key	The private half of the key pair generated in above.	String	
jira_consumer_key	The consumer key chosen in Step a above.	String	iTest_consumer
jira_access_token	Use Oauth for Zephyr for Jira to avoid exposing users' passwords. Use JIRAOAuthClient.jar to generate token If the token does not valid or expired usage: java -jar JIRAOAuthClient <the_paramter_path>. jira_access_token will be generated and write/override to the parameter file.	String	
zephyr_publish	It support for iTestUI only. When **zephyr_publish = true**, report will be upload to the sever after test case complete execution.	String	

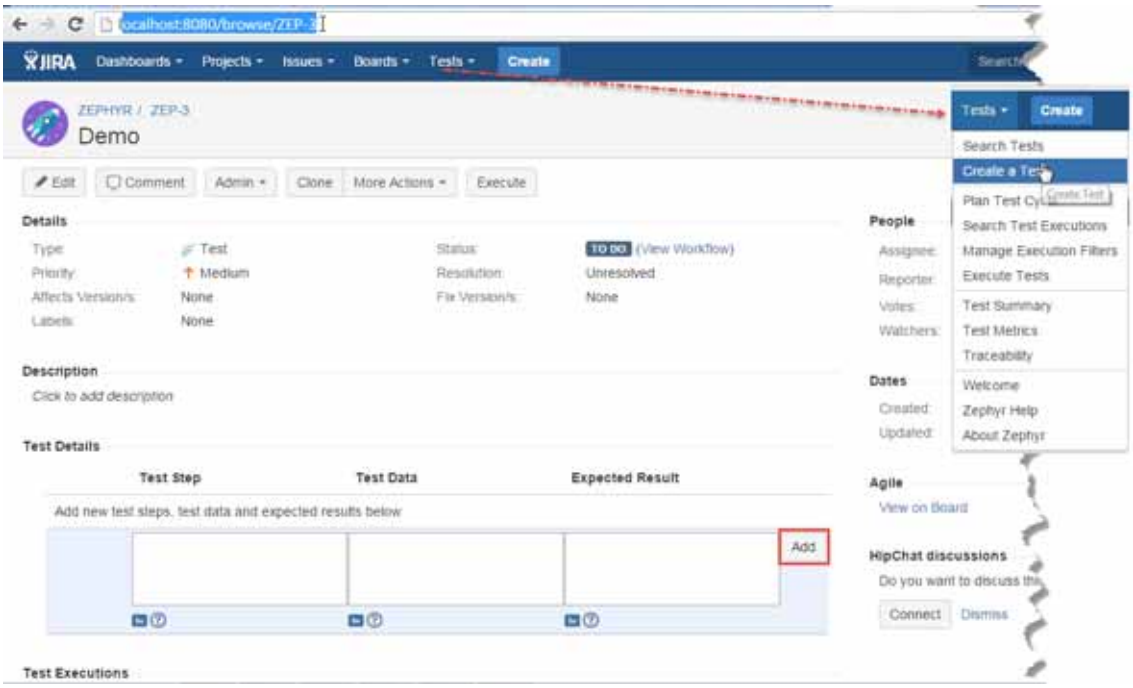
Working with Zephyr for JIRA and iTest

These topics list of tasks that describe how to use **iTest** and **Zephyr for JIRA**:

- "Creating a test case on Zephyr for JIRA" on page 1070.
- "Creating a Test Case in iTest and uploading to Zephyr for JIRA" on page 1070.
- "Verify and view the test result after execution" on page 1073.

Creating a test case on Zephyr for JIRA

1 Go to your **JIRA** and create a test.



The **Test Details** section includes the **Test Step**, **Test Data**, and **Expected Result** columns.

2 Click **Add** to create steps, the test data and the expected results manually.

- You may change the order of the steps created by dragging and dropping it in the required order.
- You may also delete a step. A message displays when you delete a step informing you that deleting the step deletes all related step information and the action cannot be reversed.

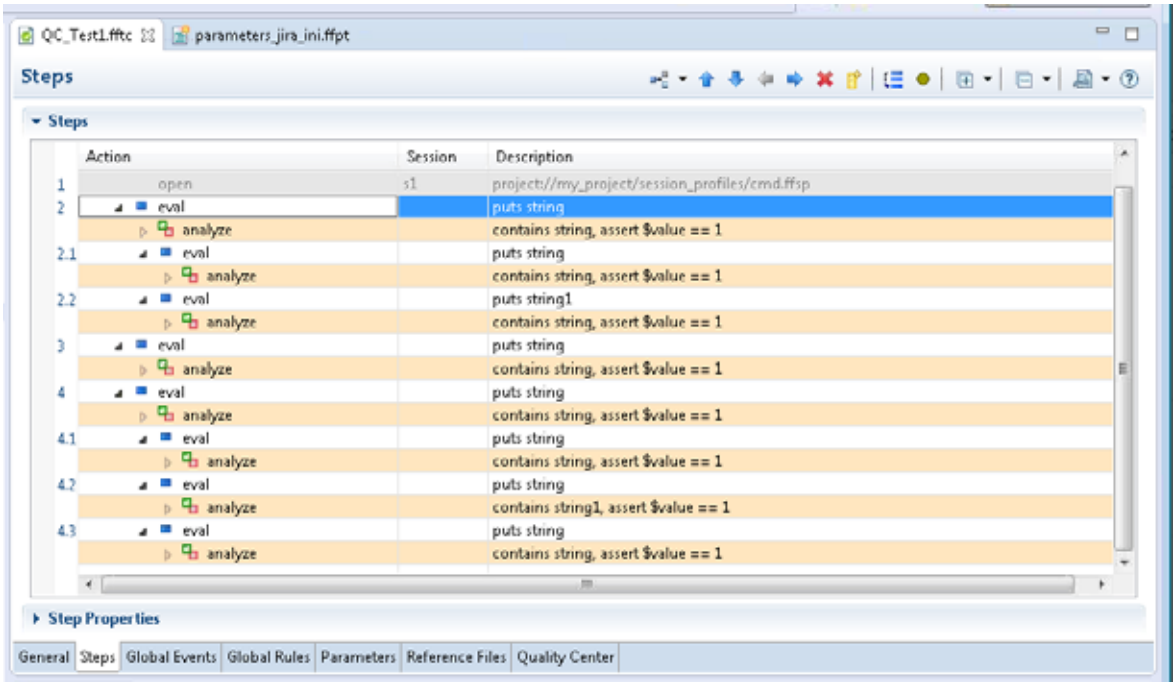
To automatically create test steps it by running iTest test case and associate JIRA test, see below, "Creating a Test Case in iTest and uploading to Zephyr for JIRA" on page 1070.

Creating a Test Case in iTest and uploading to Zephyr for JIRA

These instructions details creating an iTest Test case, p creating a parameter file to provide **Zephyr for JIRA** server information to iTest test case, and executing the iTest test case to upload test steps to **Zephyr for JIRA** test.

Step 1 Create a iTest test case

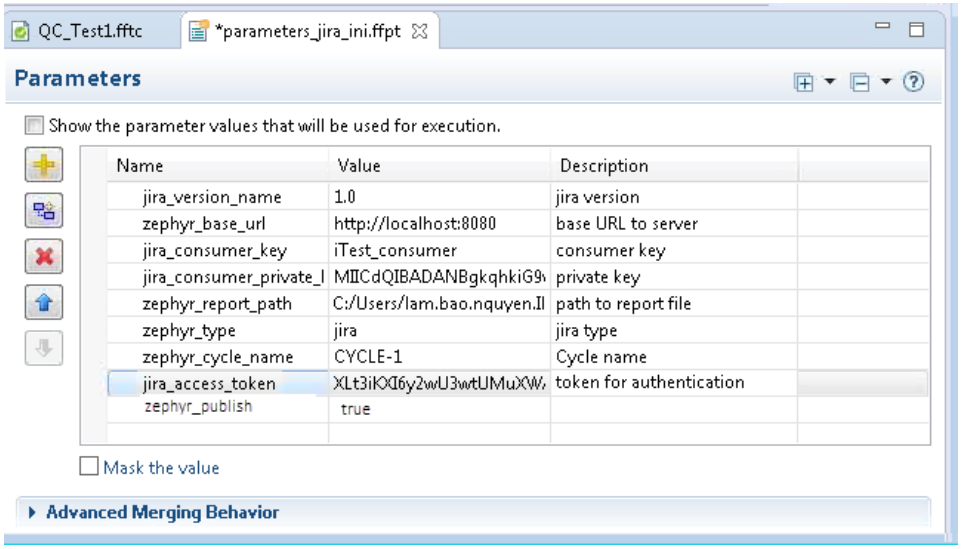
Create an iTest test case with steps you wish to share with other users within your organization.



Step 2 Create iTest parameter file with Zephyr for JIRA Information

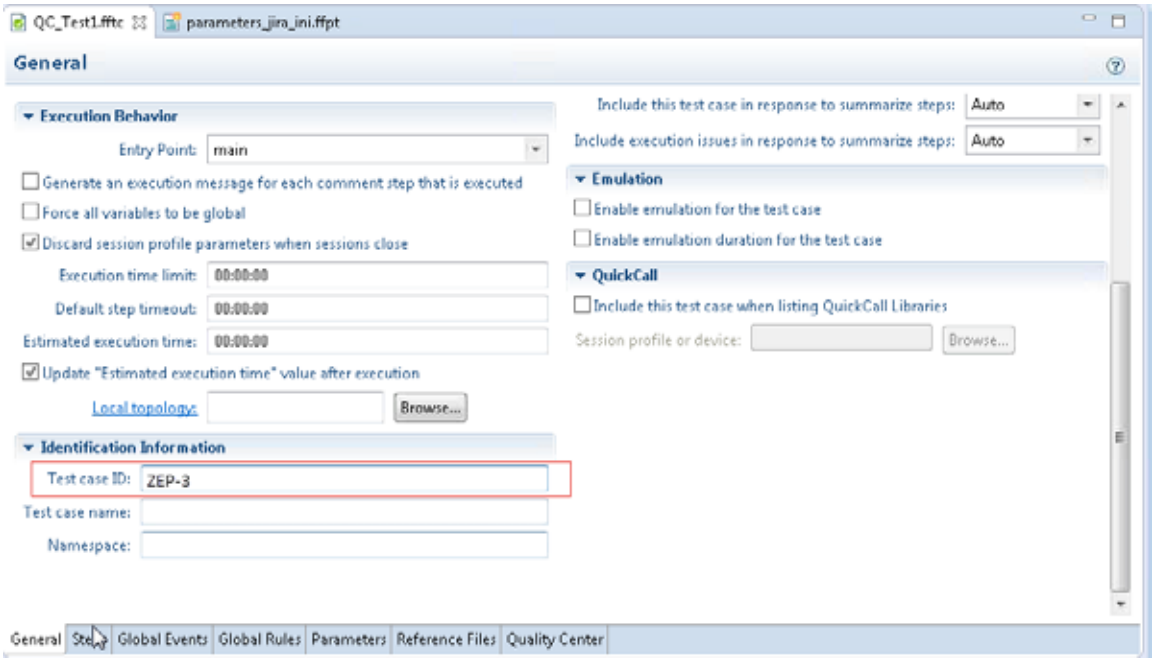
Note See "Configure the parameters file in iTest" on page 1069.

Create a parameter file with appropriate information to provide **Zephyr for JIRA** server information in the iTest test case.



Step 3 Associate iTest Test Case with Zephyr Test Case on JIRA

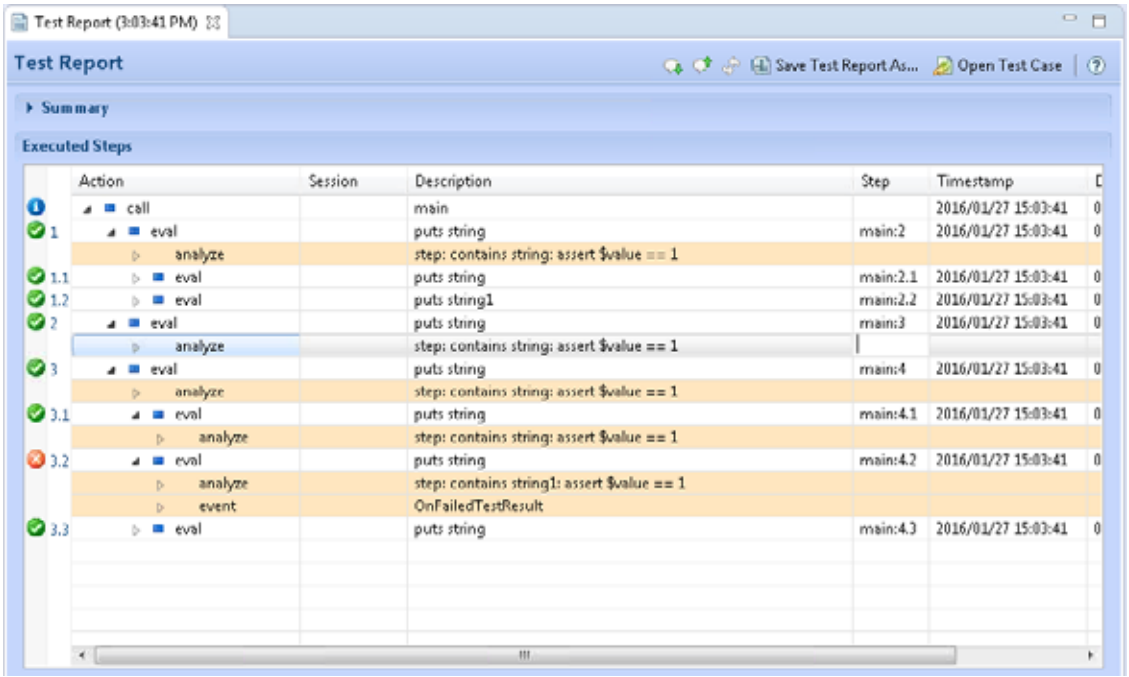
In the Test Case editor, go to the **General tab > Identification Information**. Make sure to enter the correct **Test Case ID**, that is, the name of the test you created on **JIRA**.



Note This is important for the test steps to be uploaded to the appropriate test on JIRA.

Step 4 Execute iTest test case

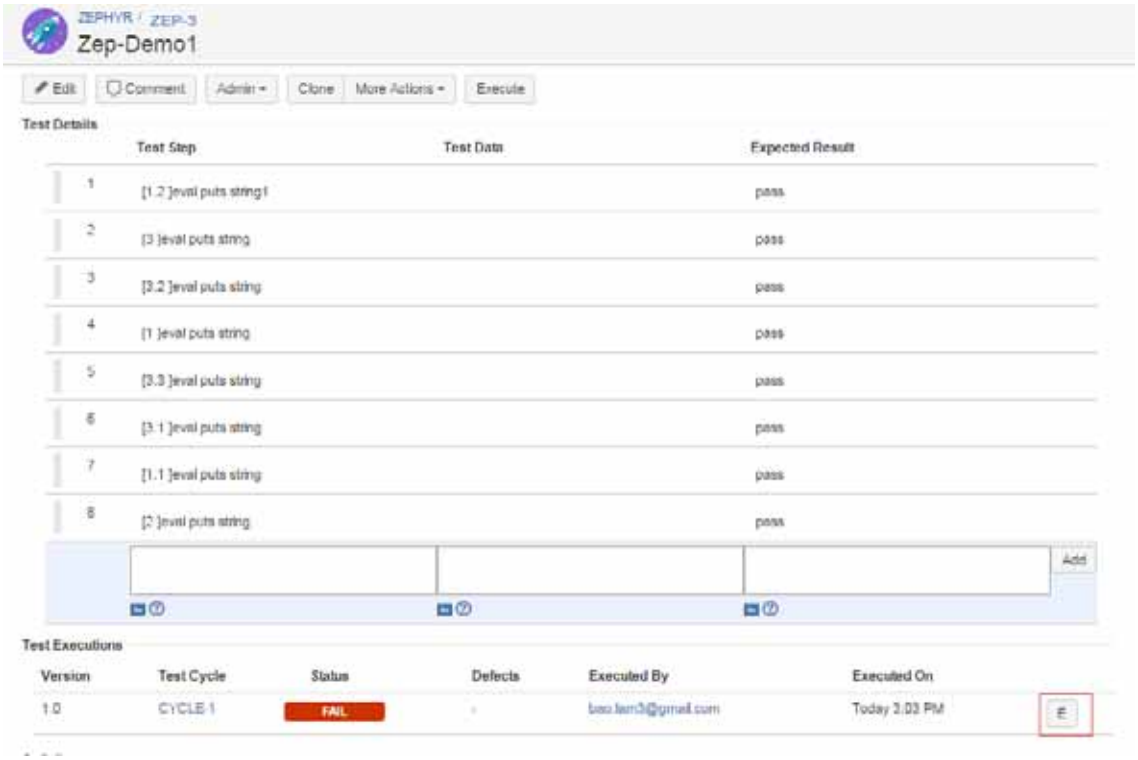
When execution completes the test steps created are uploaded to the JIRA test.



Verify and view the test result after execution

The iTest test case executed steps are uploaded to the **Zephyr for JIRA** test as a test cycle with it's status.

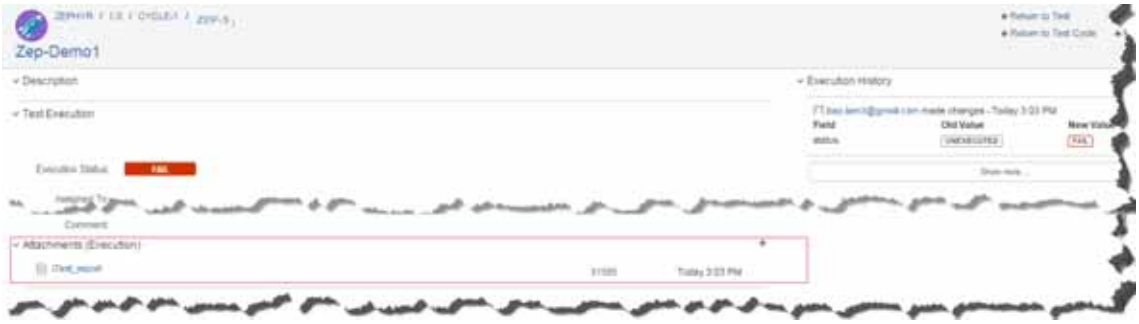
Test cycle is automatically created or updated after execution completes (the cycle name value is defined in zephyr_cycle_name parameter). If zephyr_cycle_name is not provided or is empty, the test case name will be used as Cycle name.

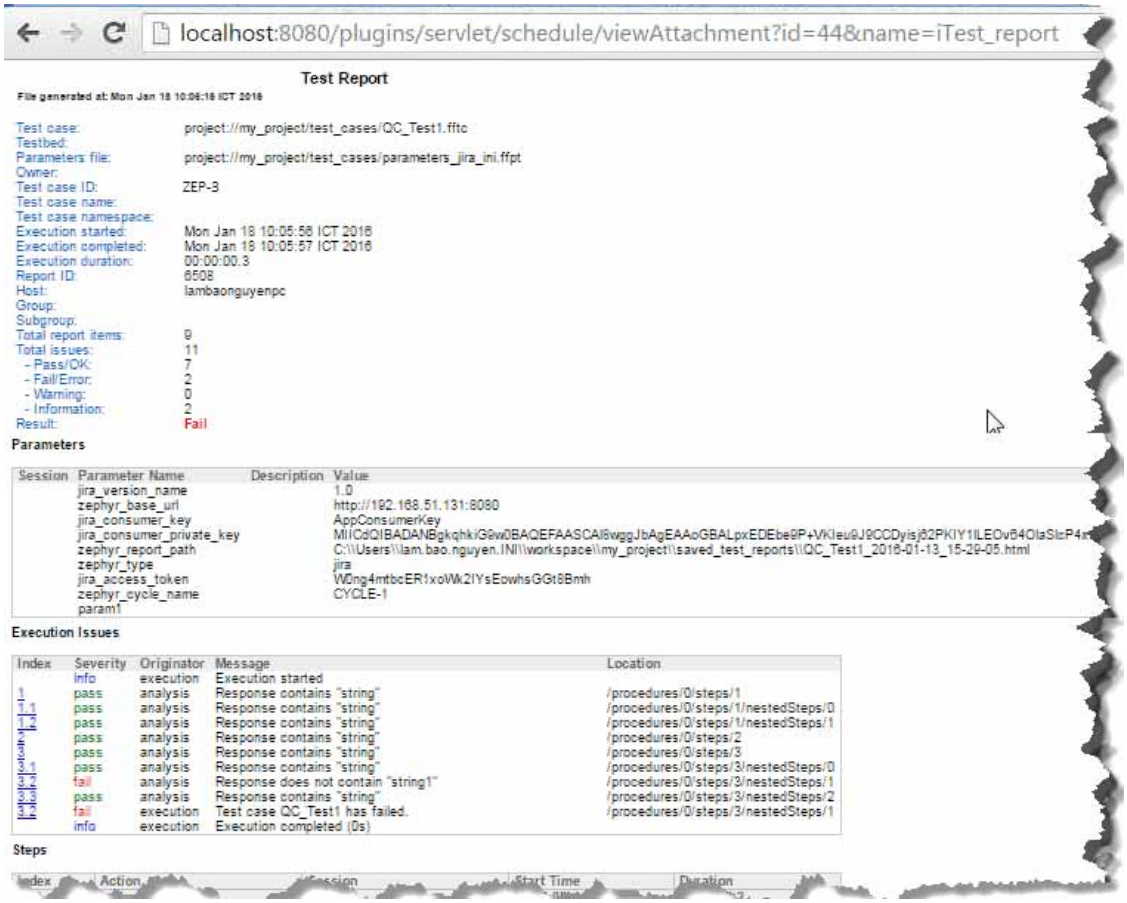


You may execute the test cycle to verify the generates test reports and perform additional task as required. That is, you may add steps, modify steps, change status, as required.

Step 1 Verify the uploaded reports

Click **E** in the **Test Executions** section to generate a report. If there are multiple Test Cycles, make sure you click **E** next to the required Test Cycle. A report generates and is attached under the **Attachments (Executions)** section. Click report to view as illustrated below.

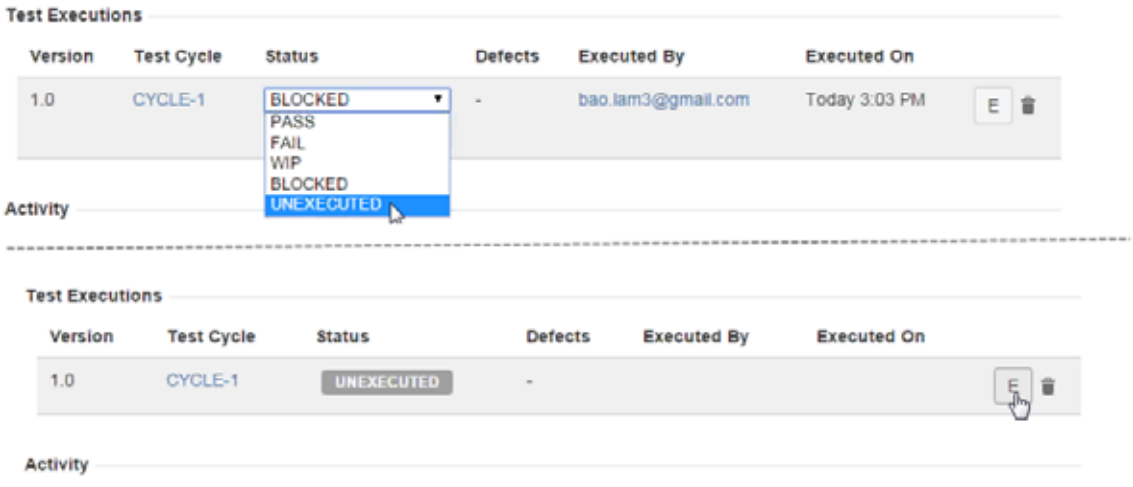




Step 2 Additional tasks in Zephyr for JIRA

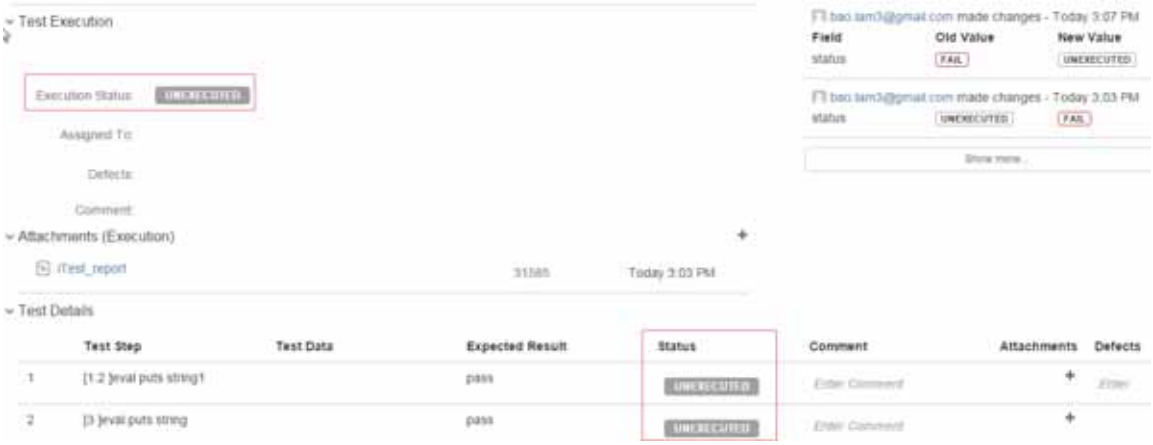
The examples below illustrates additional tasks you may perform in Zephyr for JIRA.

1 Change Status as illustrated below. The status changes, for example, **UNEXECUTED**, as you modified.



Note The information in the columns **Executed By** and **Executed On** disappears.

2 Click Execute and the **Test Details** section is updated with the **Status** you specified.

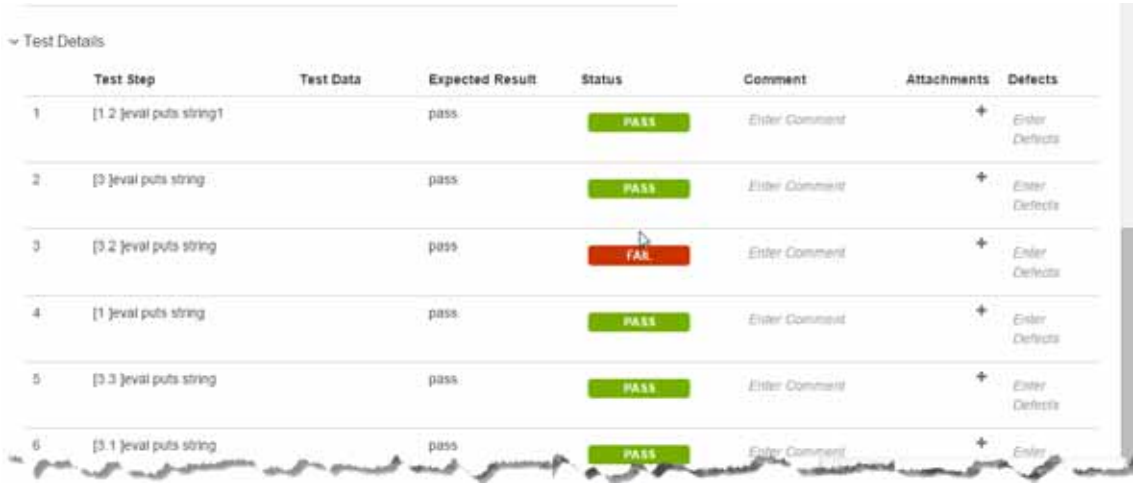


Note

- The status change history will be traced when the Execution Status is changed.
- A report also is generated and attached to the test as in Step 1 "Verify the uploaded reports" on page 1073.

3 Go to iTest and execute test case (see Step 4 "Execute iTest test case" on page 1072).

The iTest test case executed steps are uploaded to the **Zephyr for JIRA** test as a test cycle with its status updated. See illustration below.



Before publishing to Zephyr for JIRA server, iTest execution will change the Execution status of the test cycle to WIP (Work In Progress) status and then update it with the real status of execution. This is to help users have status history traced accurately for each execution in iTest.

CHAPTER 45

HTTP Sessions

HTTP session window

Using an HTTP session, a test case can talk directly with a device using the HTTP protocol operations **GET** and **POST**.

HTTP **GET** commands are useful in cases where you are not testing a Web application, but rather are testing something like a device through which the HTTP is passing.

HTTP protocol properties and options appear in the Protocol-specific property group for devices and test case steps.

Note To use Microsoft Internet Explorer 8, you must turn on Compatibility View. (Click the **Tools** button and then click **Compatibility View**.)

To use a literal IPv6 address in a URL:

- Disable field replacement (substitution) for the property.
- As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

Cookie support

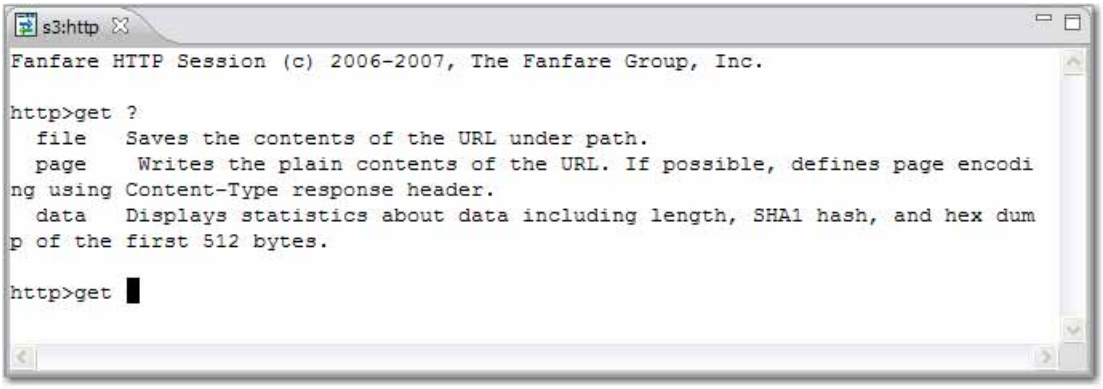
Cookies maintain stateful HTTP sessions between client and server. iTest HTTP sessions support cookies as specified in RFC 2965. iTest supports test cases that interact with a server that issues and expects to receive cookies.

HTTP command

All HTTP commands return all of the HTTP headers from the response. Use **Assert** analysis rules to analyze HTML responses.

Use the **?** character to view help for any HTTP command. For example, type **get ?** to view the

options for the **get** command.



Ending a session

Type **exit** to exit an HTTP session.

HTTP headers

You can add or override HTTP headers (for example, Content-Type or Cookie) by specifying the headers in the **Header** session profile property. The names must match exactly with the names in the source HTML. See “HTTP > Advanced Properties” on page 901.

HTTP commands

post

The **post** command supports standard HTTP **POST**. In the **Description** cell (the **Command** property), specify the URL to post to and the data to be posted.

The **post** command supports cookies as described in RFC 2965.

Syntax in test case steps

post *URL* [*timeout*] *data*

Using the post command in manual sessions

1. Type **post** *URL* [*timeout*]

   The *timeout* argument is in seconds and is optional.

2. After optionally typing the *timeout* value, press the Enter key. iTest returns a sentence about entering the data to post.

3. Type the data to post and then press the Enter key three times.

For example:

```
>post ./<Enter>
<Response:>
Your post data goes below. Use two empty lines to finalize command.
>
> my_text_to_post<Enter>
> <Enter>
> <Enter>
```

get file

get file fetches the file at the specified URL and saves it in the file system at the specified path/filename. Typically, you specify the full path.

The response in the test report contains nothing unless there is an error.

Syntax

get file *URL PathAndFilename* [*timeout*]

Note Files larger than than X MB (where X can be 100, 200, 512, etc) cannot be downloaded.

---

get page

get page fetches the contents of the specified URL and places it into the body of the response.

The **get page** command supports cookies as described in RFC 2965.

Syntax

get page *URL* [*timeout*]

get data

get data fetches the contents of the specified URL and places summarized information about it into the body of the response. The information includes:

- Length (in bytes)
- Checksum (CRC32)
- Binary dump of up to the first 512 bytes using the conventional binary decode format as shown in the example.

get data is useful for non-text files (like images) when you want to use the checksum to validate that the file has not changed from what you expected.

Syntax

get data *URL* [*timeout*]

Example get data response

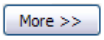
```
length 429
checksum 53C52C911F3EC9D57AD8A163929E32446DEDC179
0000 47 49 46 38 39 61 79 00 13 00 C4 00 00 DD B1 34 GIF89ay...?..??4
0010 FF BB 00 A0 A1 90 F6 B9 0D EE B6 1A 8F 9C AA 87 ??.?????.??.????
0020 99 B7 CB AD 4E 98 9E 9D E5 B4 27 B2 A5 76 D4 AF ????N?????'??v??
0030 41 A9 A3 83 C3 AA 5B 00 33 77 BA A8 69 7E 97 C4 A?????[.3w??i~??
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 21 F9 04 .............!?.
0070 00 00 00 00 00 2C 00 00 00 00 79 00 13 00 00 05 .....,....y.....
0080 FF A0 03 8D 64 69 9E 68 AA AE 6C EB BE 70 2C CF ??.?di?h??l??p,?
0090 74 6D DF 78 AE EF AC A8 0A 05 92 41 60 18 21 88 tm?x????..?A`.!?
00A0 90 02 30 29 58 8E 04 88 D1 B0 88 28 28 8D 51 9E ?.0)X?.????((?Q?
00B0 F6 66 20 04 02 0F 88 60 10 18 08 20 80 80 02 72 ?f ...?`... ??.r
00C0 00 43 1E 5F 77 A1 FC 0C 1C D0 8F 39 C3 A0 DE FA .C._w??..??9????
00D0 69 0F 04 06 0F 03 10 0B 0B 86 88 00 03 77 04 03 i........??..w..
00E0 61 0F 00 10 0C 85 0A 09 01 51 02 5F 06 00 61 07 a....?...Q._..a.
```

```
00F0 0B 0A 04 7F A3 28 3E 27 91 10 43 78 6F 92 00 0B ....?(>'?.Cxo?..
0100 04 05 03 9D 6F 09 02 81 6C 07 09 61 9A 04 91 61 ...?o..?l..a?.?a
0110 08 01 BA A4 C3 2D A8 24 B3 A8 9D 60 00 C8 5F 04 ..???-?$???`.?_.
0120 67 04 0A 07 77 9A 84 C2 68 03 45 C4 DB 28 A8 08 g...w???h.E??(?.
0130 AD 90 E0 00 BD C8 92 23 73 09 04 A2 9A 06 64 61 ???.???#s..??.da
0140 AC DC C3 A6 26 0A 03 0C 07 A2 B9 02 09 77 9D 70 ????&....??..w?p
0150 02 E5 48 D4 7B D0 20 80 92 00 6F DC C0 8B C7 90 .?H?{? ??.o?????
0160 44 9B 67 49 D2 00 08 D2 49 D3 2A 63 6C 10 41 18 D?gI?..?I?*cl.A.
0170 A0 C0 62 BB 77 18 1B 8A 1C 49 12 C7 BC 92 28 53 ??b?w..?.I.???(S
0180 2A AA 5C C9 B2 E5 96 93 2E 63 CA 9C 49 B3 E6 36 *?\?????.c??I??6
0190 1F 38 47 E4 84 B0 B3 A7 CE 9F 3C 81 FA 0C 4A 74 .8G???????<??.Jt
01A0 A8 51 A1 48 8B 26 3D AA 94 67 08 00 3B ?Q?H?&=??g..;
```

Session profile property settings for HTTP sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

HTTP

Base URL	Optional: Specify a URL to use for all HTTP commands
Authentication method	**Auto**: Determine automatically whether to use **Basic** authentication or no authentication. **Basic**: Use the specified **User ID / Password** credentials for HTTP basic access authentication. **None**: Do not perform authentication.
User ID / Password	Used when **Authentication method** is set to **Basic** or **Auto**. If the session will access a secure site, then provide the credentials that iTest should submit to gain access.

HTTP > Advanced Properties

For a session that cannot determine MIME type and character encoding from POSTs, you can use the following properties to specify the settings. You can also specify HTTP header values.

MIME type	Optional. Specify the MIME type information (for the posted content) to send when POSTing content to a server. **Note** This setting is used to tell the server the MIME type of the data. iTest does not format the data according to the MIME type that you specify. You are responsible to format the data properly. This information becomes part of the HTTP Content-Type header field for any POST or GET operation. Here is example content for the field: **Content-Type: application/x-www-form-urlencoded; charset=ISO-8859-4** Default: application/x-www-form-urlencoded
Charset	Optional. Specify the character set information (for the posted content) to send when POSTing content to a server. This information becomes part of the HTTP Content-Type header field for any POST or GET operation. Here is example content for the field: **Content-Type: application/x-www-form-urlencoded; charset=ISO-8859-4** Default : UTF-8. If UTF-8 is unavailable, then the default for the current locale is used.
Header	Optional. Specify HTTP header values, one per line, using **<header>:<spaceCharacter><value>** format. The specified values override default values ordinarily supplied by iTest. For example, with each request, iTest specifies the "User-agent" as "User-Agent: Java/1.6.0_13". You might specify a different user agent using: **User-agent: User-Agent: Java/1.6.0_14**

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

CHAPTER 55

Aptixia IxLoad sessions

IxLoad sessions and the IxLoad session window

To run an IxLoad session in iTest, you first define a test or tests in IxLoad and save the configuration file (RFX file) as you normally would. You then start the IxLoad session in iTest and run the IxLoad test. When the session starts, it checks out the libraries and connects and runs the initialization script and then loads the configuration, validates it, and transforms it (see the description for the iTest **load** command). The test produces responses and CSV files, which you can then post-process.

The IxLoad session window is an interactive terminal where you enter commands to perform IxLoad actions on the device. IxLoad returns text responses. The responses to IxLoad responses are structured and iTest uses built-in mappers to supply read-made queries.

Analyzing responses

iTest saves all responses to IxLoad commands as structured data and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. For additional information, See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.

Using Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. The intent is that you can run a session to learn more about how IxLoad sessions operate without having to install or run any Ixia software. See the descriptions of the **Demo mode** settings in the session profile: “Demo Mode” on page 1094.

IxLoad command set

This topic describes all IxLoad commands that you can submit from iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates the following queries, so you can easily work with or analyze the values of interest in the response.

- iTest provides queries by column name and elapsed time (the first column, which is always included)
- iTest also provides a query to return the list of the column names

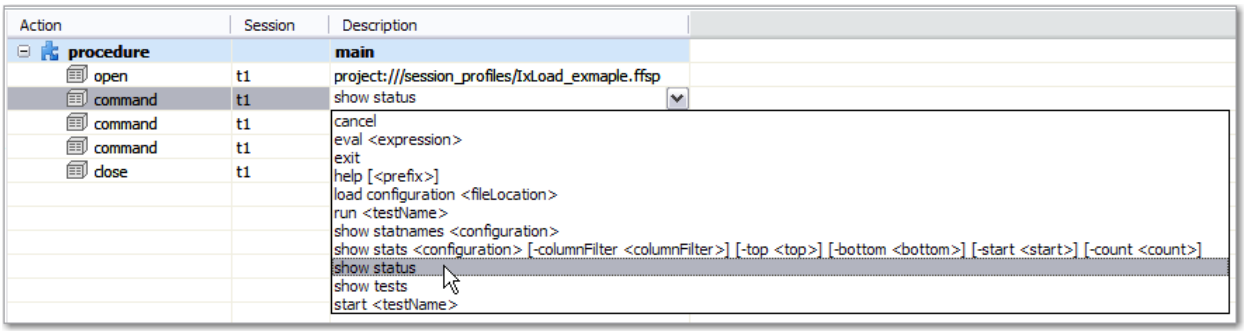
See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Properties view” on page 351, and “Structure view” on page 358.

While working in an interactive session

- Use the tab character for command completion.
- To view the list of commands while working in an interactive IxLoad session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **show stats** command, type **show stats ?**.

While working on a test case in the Test Case editor

1. For a step in an IxLoad session, in the **Action** cell, select **command**.
2. Select the command from the drop-down list in the **Description** cell.



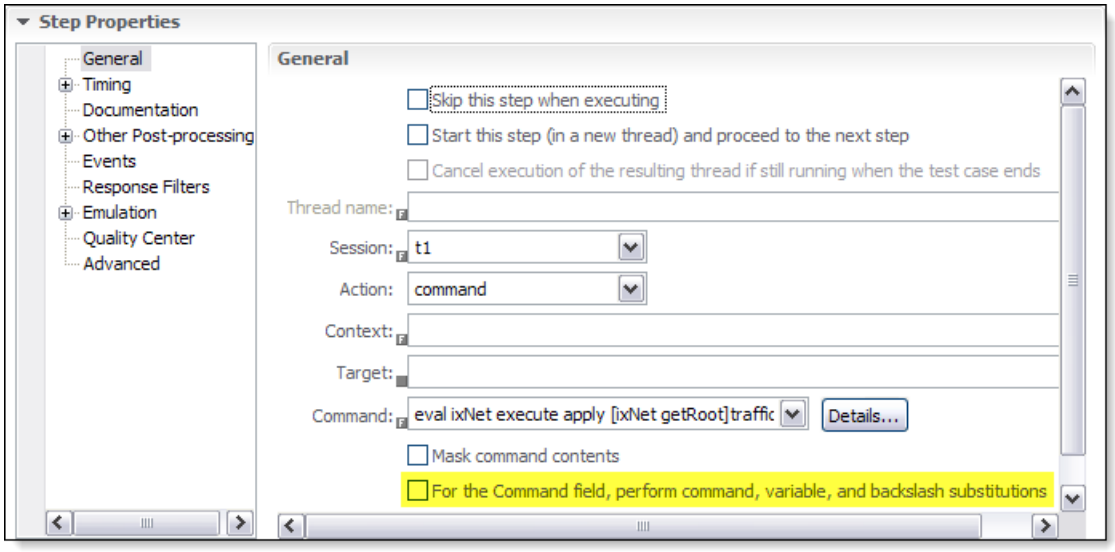
Perform substitution on Ixia commands

Test case steps that use the iTest **command** action to perform Ixia commands can include proper Ixia syntax that would be incorrectly substituted by the iTest interpreter. Here is an example Ixia command that applies traffic items onto the configuration in Tcl:

eval ixNet execute apply [ixNet getRoot]traffic::ixNet::OK

By default, the iTest interpreter would read **[ixNet getRoot]** as something to substitute and then fail while trying to execute a (nonexistent) iTest **ixNet** command.

For this reason, be sure to configure each **command** step to ***not*** perform command substitution: In the step properties, uncheck the **For the Command field, perform command, variable, and backslash substitutions** checkbox.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

IxLoad command set

cancel	Cancels the current execution.
exit	Closes the session. If an execution is underway, a message notifies you that execution is being canceled.
help [*prefix*]	Displays the list of IxLoad commands with brief descriptions.
load configuration {*URI* | *path*}	Loads a new RXF configuration file (*.rxf). The file may be inside or outside of the workspace. You must generate RXF files using IxLoad. If you use a conventional path rather than a URI, the configuration file loads and then: iTest creates a temporary configuration file and substitutes the port number information from the session properties (unless no ports in the client/server category were provided). iTest uses the **Client port list** property to replace card and port tags in the Client Network section — replacing in the same order as the item tags appear in the RXF file. The same process occurs for the server side using the **Server port list** property. The RXF file will not be modified. See the descriptions of the **Client port list**, **Server port list, RXF ports,** and **Mapped ports** properties in “Session profile property settings for IxLoad sessions” on page 1089. **Notes**: iTest does not support configurations involving multiple chassis. You can perform the **load configuration** -> **run** workflow multiple times in the same session.
run [*testName*]	Runs the specified test in the current configuration to completion (synchronous execution). You can start only one test at a time. (For asynchronous execution, see the **start** command.) If **testName** is not specified, then the command runs all tests in the configuration. Type **show stats** to see results You can cancel the **run** command using Ctrl-C — the response includes any results up to that point.

start [*testName*]	Starts the current configuration running in the background (asynchronous execution). You can start only one test at a time. **Note** Manual testing only: Because **start** is an asynchronous command, you may get unexpected results if you immediately submit another command (for example **show status**) immediately after a **start** command. In contrast, the **start** / **show status** sequence is completed correctly in automated iTest test cases. For synchronous execution, use the **run** command. If *testName* is not specified, then the **start** command starts all tests in the configuration, one at a time. Type **show stats** to see results You can cancel the **start** command using Ctrl-C and the response includes any results up to that point.
show stats *resultsFile* [**-stat** *statPattern*] [**-columnFilter** *column_Filter*] [**-top** *count*] [**-bottom** *count*] [**-start** *startRow*] [**-count** *countOfRows*]	Dumps all or a subset of one of the log files generated as a result of a run. *resultsFile:* Specifies which of the log files to display. (The IxLoad User's Guide explains how log file are named.) You can use command completion (tab and ?) to view the available logs. iTest does not display the timestamp and the filename extension (for example, if the filename is **TelnetClient_200806131545889.csv**, iTest displays **TelnetClient** when you type “?" at the command prompt. If you specify the results folder, then iTest does not delete the files when the session ends. **-columnFilter**: Specify a list of column names. Only the specified columns appear in the displayed response. The filter supports the "*" wild card. **-start**: Enables you to extract data other than at the start and end of table. If you specify **-start**, then you must also specify **-count.** **-count**: Specifies the number of rows to retrieve. Required only if **-start** is specified. **Note**: You can specify EITHER (**-start** and **-count**) OR (**-top** and/or **-bottom**). If you do not specify any of the flags, iTest uses the default values of 25 for **top**, and 25 for **bottom**. **-top**: The table shows the top <count> rows from the CSV file. If omitted, default is 25. **-bottom**: The table shows the bottom <count> rows from the CSV file. If omitted, default is 25.

show status	Displays the current status — either “Not running” or “Running”. Manual testing only: Because **start** is an asynchronous command, you may get unexpected results if you immediately submit another command (for example **show status**) immediately after a **start** command. In contrast, the **start** / **show status** sequence is completed correctly in automated iTest test cases. For synchronous execution, use the **run** command.
show tests	Lists all tests defined in the current configuration.

Session profile property settings for IxLoad sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

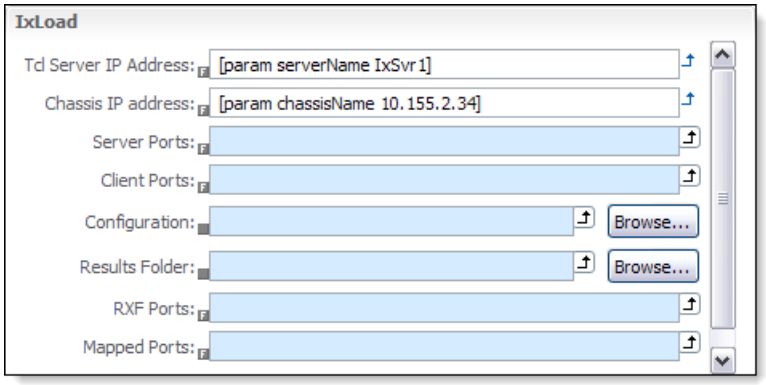
Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Using Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. You do not need to configure or connect to an IxLoad session to run in demo mode. See the descriptions of the **Demo mode** settings in the session profile: “Demo Mode” on page 1094.

IxLoad



Tcl server IP address	Required on Linux. Specify the IP address or hostname of the device. Typically only used on Unix computers to point to an intermediate Windows computer that runs the Ixia Tcl Server. In the example, the **Tcl server IP address** property value is specified dynamically at runtime by using a param command to pass the **serverName** parameter.
Chassis IP address	Required. Specify the IP address or hostname of the chassis to connect to. In the example, the **Chassis IP address** property value is specified dynamically at runtime by using a param command to pass the **chassisName** parameter.
Client port list	Optional. Specify a list of ports using Mapped Ports to replace card and port tags in the **ClientNetwork** section of the RFX file. The ports are replaced in the order of the item tags in the RXF file. The number of ports in the list must be the same as the number of ports in the RXF file. If you do not specify a list of ports for this property, iTest uses the ports listed in the RXF configuration file. **Note** The Client port list is not replaced as specified in the RXF file if "RFX ports" and the "Mapped ports" fields are empty. **Syntax**: Use <card_a>:<ports>,<card_b>:<ports> syntax. You can use the * wildcard character to indicate all cards. **Examples**: To specify card 2, port 1, use 2:1 To specify multiple cards and ports, use 2:1,3:2,4:3
Server port list	Optional. Specify a list of ports using Mapped Ports to replace card and port tags in the **ServerNetwork** section of the RFX file. The ports are replaced in the order of the item tags in the RXF file. **Note** The Server port list is not replaced as specified in the RXF file if "RFX ports" and the "Mapped ports" fields are empty. The number of ports in the list must be the same as the number of ports in the RXF file. If you do not specify a list of ports for this property, iTest uses the ports listed in the RXF configuration file. **Syntax**: Use <card_a>:<ports>,<card_b>:<ports> syntax. You can use the * wildcard character to indicate all cards. **Examples**: To specify card 2, port 1, use 2:1 To specify multiple cards and ports, use 2:1,3:2,4:3
Configuration	Optional. Specify a URI pointing to the RXF file that was generated by IxLoad (the file can be either inside or outside the workspace). The RXF file resides on the Windows computer that syringing the TCL server. If you provide a conventional path string, iTest converts it into a URI.
Results folder	Optional. Specify the URI of the folder where all results files should be written. The folder can be either inside or outside the workspace. For example, **file:///c:/temp/dir** **For Linux**: Results folder is a path on the Windows computer that is running the TCL server. If you run tests on Linux or Solaris, then you should manually clean up the folder after testing.

RXF ports	Optional. If you do not specify a value for **RXF ports** or **Mapped ports**, then the session uses the ports for Traffic Flows as they appear in the RXF file. If you specify one or more **RXF ports**, then you must specify a corresponding port in the **Mapped ports** property. Provide a comma-separated list. iTest searches the RXF file for a port in the **RXF ports** list and replaces it with its corresponding Mapped port. The number of entries in the **RXF ports** and **Mapped ports** text boxes must match. **Note** The Server port list and Client port list are not replaced as specified in the RXF file if "RFX ports" and the "Mapped ports" fields are empty.
Mapped ports	Optional. See the **RFX ports** property.
Force to take ownership	When you select this option, upon connecting, the session takes ownership of the ports even if another user currently has ownership. **Note** You do not have to go to Ixia GUI to clear port owner to make sure that the ports are available for your use.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX X.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Tcl Interpreter

Ixia sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the session profile, or you can set a preference for all Ixia sessions. You can specify the path to the correct interpreter here, in the session profile. Refer to Ixia documentation for additional Tcl version information.

For example, if Ixia N2X software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the Ixia N2X session. If, instead, you change the preference for all Ixia N2X sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Ixia N2X sessions, use the Preference page. See “Session profile property settings for Ixia N2X sessions” on page 1114.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Demo Mode

Run session in demo mode	Check the box to run the session in “demo mode”, in which each IxLoad command returns a fixed response that mimics a real response from an Ixia device. The intent is that you can run a session to learn more about how IxLoad sessions operate without having to install or run any Ixia software. The responses are mapped by built-in iTest mappers to generate usable structured data. When running in demo mode, the session window tab title will end with **(demo)**, for example, **s2:IxLoad (demo)**.

Example IxLoad session text

```
Spirent IxLoad command interpreter. Copyright (c) 2005 - 2012, Spirent
Communications, Inc.


Using external tcl interpreter
tcl version: 8.4.13
tclsh location: C:/Tcl/bin/tclsh.exe
tcl library: C:/Tcl/lib/tcl8.4
tcl package search path: {C:/Program Files/Spirent Communications/Spirent
TestCenter 3.30/Layer 4-7 Application/TclAPI} {C:/Program Files/Spirent/iTest
3.4} C:/Tcl/lib/tcl8.4 C:/Tcl/lib


Running initialization script...

Loading TCL helper scripts...OK

Loading TCL package 'IxLoad'...OK

IxLoad version being used: 4.10.94.84
Please ensure that you are using the compatible IxOS version: 5.30 GA

Connecting to remote TCL server at ectclsvr1...OK

Loading repository file.....OK

C:\DOCUME~1\hku\LOCALS~1\Temp\simpleHTTP_AddMorePageRequests_4.10GA2041335240
791499388.rxf

SYNC: client network @ Wed Apr 28 11:14:33 2010
SYNC: server network @ Wed Apr 28 11:14:34 2010
*Debug 04/28 11:14:35 ixLogger: Current Date is 04/28/2010
*Info 04/28 11:14:35 Chassis: Connecting to chassis 10.155.2.33
*Info 04/28 11:14:46 Chassis Chain: Connected successfully to the chassis
10.155.2.33
*Info 04/28 11:14:46 Chassis: Connecting to chassis 10.155.2.34
*Info 04/28 11:14:51 Chassis Chain: Connected successfully to the chassis
10.155.2.34

Configuration loaded successfully
Ixia IxLoad>?
  cancel - Cancels current test execution
  eval   - Evaluate Tcl expression
  exit   - Exit the application
  help   - Display command help information
  load   - Load new active configuration file
```

```
   run   - Runs the specified test in the current configuration to completion
   show  - Display information
   start - Starts the current configuration running in the background

Ixia IxLoad>show ?
   statnames - list of available statistics
   stats   - Display statistics table
   status  - Show current running status
   tests   - Show list of tests available in the current configuration

Ixia IxLoad>show stats ?
   <configuration> - name of existing statistic file
   -columnFilter  - list of statistic columns pattern
   -top       - number of top elements
   -bottom     - number of bottom elements
   -start     - start index of rows
   -count     - number of elements after start index
   Test_Client
   Test_Server

Ixia IxLoad>
```

Setting preferences for Ixia sessions

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Session Types > Ixia <type>**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if Ixia Traffic software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the Ixia Traffic session. By changing the preference under **Session Types > Ixia Traffic**, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case

to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: Auto-select
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked
Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

CHAPTER 56

Aptixia IxNetwork sessions

IxNetwork session window

In IxNetwork sessions, you can start and stop traffic, start and stop capture, and review statistics.

The levels of the Object Data Matrix are represented as subdirectories (see Table 1-11, API Command Data Model Structure in the *IxNetwork Tcl API Guide*). The iTest IxNetwork session window enables you to navigate the object hierarchy by using commands that you typically use to navigate a file system.

If you think of the objects in the hierarchy as leaves and branches of a structured tree, it is natural to use file system commands to “move around” in the tree. For example, **cd ..** moves up one level in the hierarchy, **cd /** moves to the root.

Commands are sensitive to the context of the "current working directory". You can determine the "current working directory" (current object) at any time using the **pwd** command. **ls** and **dir** (identical commands) return the elements in the "current working directory".

See “Example IxNetwork session” on page 1106.

Analyzing responses

iTest auto-maps IxNetwork responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries.

See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Properties view” on page 351, and “Structure view” on page 358.

Starting an IxNetwork session

You have the option to specify a configuration file in the **Configuration script** property in the session profile. You use IxNetwork to configure ports and then save the configuration to a binary file as you normally would.

When you start the IxNetwork session in iTest, iTest checks out the libraries and connects and runs the configuration script (if specified) and the initialization script (if specified) and then loads and validates the configuration.

About your installation

Note You have the option to run IxNetwork sessions in demo mode — no Ixia software required. The intent is that you can run a session to learn more about how IxNetwork sessions operate without having to install or run any Ixia software. See "Session profile property settings for IxNetwork sessions" on page 1100.

You must run IxNetwork Tcl server on the computer that is running iTest. iTest executes the IxNetwork client. You must run the identical version of IxNetwork on the server, client, and chassis.

When iTest submits an IxNetwork command, the IxNetwork Tcl server window returns to the system tray. It is this instance of IxNetwork whose configuration is saved.

When the configuration script is evaluated, this instance is configured. In other words, if you are working with iTest and IxNetwork, you only want to use the instance of IxNetwork that was launched with the IxNetwork Tcl Server shortcut. To open the IxNetwork user interface, double-click the icon in the system tray. The text (TCL:8009) appears in the IxNetwork window title.

Using Demo mode

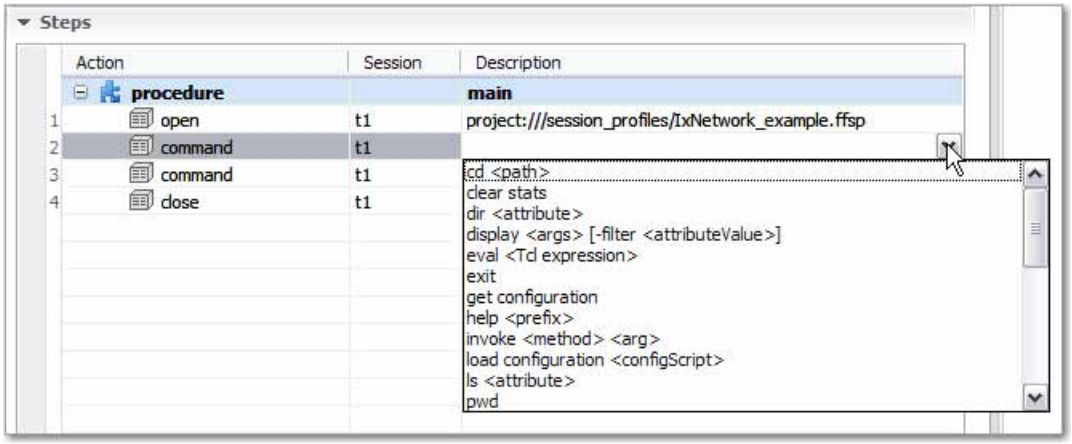
You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. You do not need to configure or connect to an IxNetwork session to run in demo mode. See the descriptions of the **Demo mode** settings in the session profile: "Demo Mode" on page 1105.

IxNetwork commands

See Ixia documentation for command descriptions.

While working on a test case in the Test Case editor

1. For a step in an IxNetwork session, in the **Action** cell, select **command**.
2. Select the command from the drop-down list in the **Description** cell.



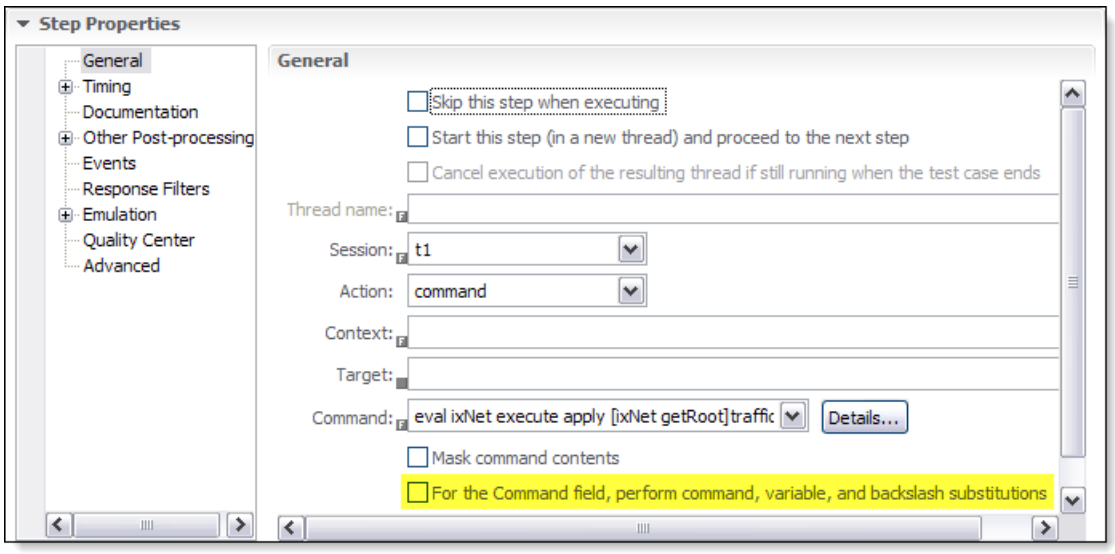
- **Perform substitution on Ixia commands**

  Test case steps that use the iTest **command** action to perform Ixia commands can include proper Ixia syntax that would be incorrectly substituted by the iTest interpreter. Here is an example Ixia command that applies traffic items onto the configuration in Tcl:

  eval ixNet execute apply [ixNet getRoot]traffic::ixNet::OK

  By default, the iTest interpreter would read **[ixNet getRoot]** as something to substitute and then fail while trying to execute a (nonexistent) iTest **ixNet** command.

  For this reason, be sure to configure each **command** step to ***not*** perform command substitution: In the step properties, uncheck the **For the Command field, perform command, variable, and backslash substitutions** checkbox.



Session profile property settings for IxNetwork sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 120. For a detailed description of how iTest uses the settings, see "About property settings" on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Using Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. See the descriptions of the **Demo mode** settings in the session profile: "Demo Mode" on page 1105.

IxNetwork

Note You can configure Tcl server settings for all IxNetwork sessions. See "Setting preferences for Ixia sessions" on page 1105.

Tcl Server	Required. Specify the IP address or hostname of the IxNetwork Tcl server. In the example, the **Tcl server** property value is specified dynamically at runtime by using a param command to pass the **serverName** parameter. Defaults: Windows: localhost Linux: [blank]
Tcl Server Port	Required. Specify the IP port. Default: 8009
Tcl Server Username	Enter the server user name of the IxNetwork instance used for the IxNetwork Session. **Important** Make sure that the IxNetwork Connection Manager is running with the instance of the IxNetwork server user name. **Close on disconnect:** Select to indicate that the IxNetwork session will be terminated after the automation session ends. When **Close on disconnect** is *not selected*, the IxNetwork session will not terminate even after the automation session ends. You will have to enter explicit command to stop any traffic on the network and close session. **Note** When you **Start** the IxNetwork session, you will notice that the Ixia Connection Manager connects to the server instance name specified and automatically opens the IxNetwork API Server.
Chassis	Required. Specify the IP address or hostname of the Ixia device to connect to. In the example, the **Chassis** property value is specified dynamically at runtime by using a param command to pass the **chassisName** parameter.

Ports	Specify the ports to use for the session. Use **<card_a>:<ports>,<card_b>:<ports>** syntax. You can use the ***** wildcard character to indicate all cards. **Note** To specify all ports on a card, use the **Cards** property.You typically supply a value for either the **Cards** property or the **Ports** property. **Examples** To specify card 2, port 1, use **2:1** To specify multiple cards and ports, use **2:1,3:2,4:3** **Note** iTest uses the ports in the order specified in the session profile. For example, If the session profile specifies **1:2,1:3**, and you save the Ixia config file using only one port, then configuration load command will always pick **1:2** as the port for loading. If you want to load **1:3**, then you need either to change the order of the ports in the session profile, or to create a new session profile that specifies just **1:3**.
Configuration script	Optional. Specify either a binary configuration file or a TCL configuration file. Specify a URL pointing to the binary file that was generated by IxNetwork (the file can be either inside or outside the workspace). As a result, when the session starts, the device is configured exactly as if you had configured it using IxNetwork. The Browse button shows only binary files, but you can specify a TCL file using a path. If you provide a conventional path string, iTest converts it into a URI. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
Force to take ownership	When you check the box: Upon connecting, the session takes ownership of the ports even if another user currently has ownership. Default: unchecked
Validate chassis and ports upon start of session	When you check the box: If the configuration file is omitted, then IxNetwork will verify that the Tcl server is connected to the specified chassis on the specified ports. Default: unchecked
Start IxNetwork Tcl server if it is not running	When you check the box: Upon starting the session, determine whether the server that is specified in the **Tcl server** property is running, and if not, start it before connecting. Default: unchecked

Tcl server connect delay	Specify a delay in integer seconds between the first IxNetwork Tcl server response after the session starts and the attempt by iTest to connect to the server. The default 45 second delay is typically effective. **Note** This delay is a workaround to a known issue with IxNetwork 5.50 and 5.60: If iTest attempts to connect to the Tcl Server too quickly after the server first responds, then the connection attempt will hang. The delay is used only if all of the following are true: • The IxNetwork Tcl server is not running • The **Start IxNetwork Tcl server if it is not running** checkbox is checked • The **IxNetwork API version** property is set to 5.50 or 5.60
Release ports and clear configuration when the session ends	If unchecked (default), then ports are not released and are left as currently configured when the session ends. This setting is useful for reducing test execution time when running a batch of IxNetwork tests. Releasing port ownership can cause ports to be rebooted, which can take a significant time. If checked, then iTest clears port ownership and configuration when the session ends. Default: unchecked
IxNetwork API version	Specify the version of IxNetwork Tcl APIs to use. This setting ensures that iTest sends the proper commands to IxNetwork. Default: 5.30

IxNetwork >Timeouts

Display command timeout	Timeout for return of response to command. Default: 5 seconds
Statistics timeout	Timeout for return of response to request for statistics. Default: 5 seconds
Chassis Connection timeout	**Note** This setting is used only for versions of IxNetwork that have the bug that caused the IxNetwork Tcl server to not return a success message upon connecting after repeated requests to connect (from the iTest session). When iTest notices that the IxNetwork Tcl server is not running, it auto-starts the server. This timeout allows sufficient time for the server to start and return a success message.

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Tcl Interpreter

Ixia sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the

session profile, or you can set a preference for all Ixia sessions. You can specify the path to the correct interpreter here, in the session profile. Refer to Ixia documentation for additional Tcl version information.

For example, if Ixia N2X software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the Ixia N2X session. If, instead, you change the preference for all Ixia N2X sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Ixia N2X sessions, use the Preference page. See "Session profile property settings for Ixia N2X sessions" on page 1114.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Demo Mode

Run session in demo mode	Check the box to run the session in "demo mode", in which each IxNetwork command returns a fixed response that mimics a real response from an Ixia device. The intent is that you can run a session to learn more about how IxNetwork sessions operate without having to install or run any Ixia software. The responses are mapped by built-in iTest mappers to generate usable structured data. When running in demo mode, the session window tab title will end with **(demo)**, for example, **s2:IxNetwork (demo)**.

Setting preferences for Ixia sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Ixia <type>.**

General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if Ixia Traffic software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the Ixia Traffic session. By changing the preference under **Session Types > Ixia Traffic**, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: Auto-select
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked
Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

Prompt Style

Prompt style	Specify the type of prompt to expect for IxNetwork sessions. Compressed path Full path Path suffix only No path

Example IxNetwork session

Example use of the **startCapture** and **saveCapture** commands. Annotations appear in blue next to the command line.

```
Using external Tcl interpreter
Tcl version: 8.4.15
Tcl location: /usr/bin/tclsh
Tcl library: /usr/lib/tcl8.4
Tcl package search path: /home/komaz/Ixia/lib /home/komaz/Smartbits/bin
```

```
/home/komaz/Spirent_TestCenter_2.01/Spirent_TestCenter_Application_Linux
/home/komaz/Agilent/N2X/tcl/extras /home/komaz/Ixia/lib/IxTclNetwork
/usr/lib/tcl8.4 /usr/lib


Loading Tcl package IxTclNetwork... OK (version 5.30)

Connecting to Tcl Server at 192.168.3.139:8009... OK (version 5.30.40.59)

IxNetwork /> load configuration project:///programs.ixncfg
Loading configuration file
/home/komaz/work/itest/my_project/programs.ixncfg...
    Cleaning up existing configuration... OK
    Loading configuration file... OK
    Verifying port count... OK
    Assigning physical ports... OK
    Applying configuration... OK
Configuration successfully loaded from
/home/komaz/work/itest/my_project/programs.ixncfg

IxNetwork /> start capture
Port 'Ethernet - 001' is in improper rxMode. Skipped.
Port 'Ethernet - 002' is in improper rxMode. Skipped.
```

< I want to start capture on this port

```
Successfully started capture

IxNetwork /> cd vport:2
```

< So I'm cd'ing to this port and set the rxMode attribute to the correct value

```
IxNetwork /vport:2> start capture
Port 'Ethernet - 001' is in improper rxMode. Skipped.
Neither Data or Control capture is enabled for port 'Ethernet - 002'. Skipped.
```

< I also need to enable capture for this port

```
Successfully started capture

IxNetwork /vport:2> cd capture
```

< cd'ing to capture object and enabling data capture

```
IxNetwork /vport:2/capture> start capture
Port 'Ethernet - 001' is in improper rxMode. Skipped.
Starting capture on port 'Ethernet - 002'  .OK
```

< successfully started

```
Successfully started capture

IxNetwork /vport:2/capture> start protocols
```

< generate some traffic

```
..............
Successfully started protocols on ports 4:1,4:2

IxNetwork /vport:2/capture> start traffic
Applying traffic  ....OK
Starting traffic  OK
Successfully started traffic

IxNetwork /vport:2/capture> stop traffic
Stopping traffic  OK
Successfully stopped traffic

IxNetwork /vport:2/capture> stop capture
```

```
Stopping capture on port 'Ethernet - 001'  OK
Stopping capture on port 'Ethernet - 002'  ..........OK
```

< stopping capture

```
Successfully stopped capture

IxNetwork /vport:2/capture> save capture 4:2 project:///capture.cap
```

< Saving capture. Capture will be saved in the capture_Data.cap file. If Control capture (softwareEnabled attribute) is configured, then it will be saved in the capture_Control.cap file.

CHAPTER 57

Ixia N2X sessions (Obsolete and Deprecated)

Important
The Ixia NX2 sessions are ***obsolete and is no longer supported*.**

This chapter is provided only to support existing implementations.

Ixia N2X session window

The Ixia N2X session window is an interactive terminal where you enter commands to perform Ixia N2X actions on the device. Ixia N2X returns text responses.

Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Ixia devices. iTest captures both the text response and a structured version of the response. iTest auto-generates appropriate queries for response data, so you can easily work with or analyze data of interest in the response. In an interactive Ixia N2X session, you can:

- Get the current configuration settings from an Ixia device. The **configuration save** command requests the current configuration settings of the device. The settings are returned in the response as a Tcl script. The **configuration get** command requests the same information and displays it on the terminal.

  Note Both **configuration save** and **configuration get** perform replacements on the file so that device-specific information (like IP address and port numbers) are translated into variables or command substitutions.

- You can use the **configuration load** command to load the configuration settings (from the response to the **configuration save** command) into the device before performing other steps.
- Request and display statistics, start and stop traffic, start and stop collisions, and perform other functions commonly used during a test.

iTest auto-maps N2X responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.

Tip When you close a session with an **exit** command, a **close** step is captured and (if so configured) the session window remains open. (To configure the session window to remain open after you disconnect a session, set the following two **Execution** preferences appropriately: **When execution finishes, close all Sessions windows** and **Before executing, close all active and inactive session windows.** See “Setting preferences for execution” on page 330.)

Interactive Ixia N2X command set

The list of commands appears in “Ixia N2X command set” on page 1111.

- To view the list of commands while working in an Ixia N2X session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **stream send** command, type **stream send ?**
- Slot and port numbering starts with zero (zero-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.
- Use the * wildcard character to represent all slots or ports

Ixia N2X command set

This topic describes all Ixia N2X commands that you can submit from iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or apply analysis rules to the values of interest in the response.

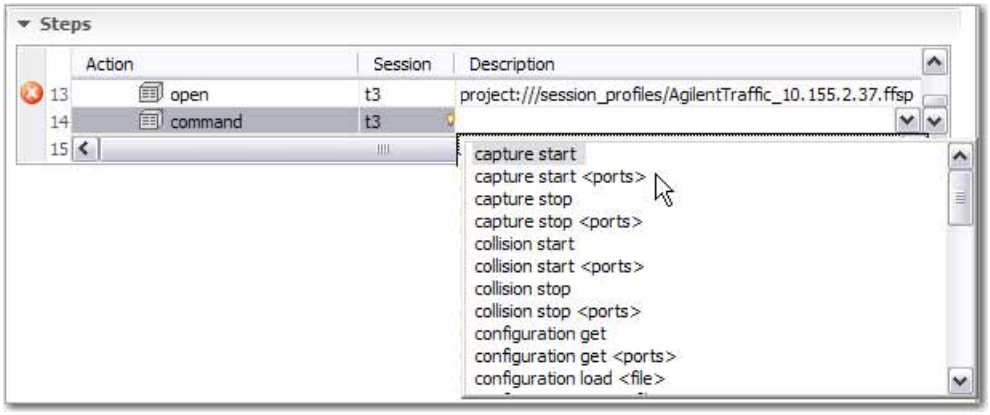
While working in an interactive session

- To view the list of commands while working in an interactive Ixia N2X session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **stream send** command, type **stream send ?**.

While working on a test case in the Test Case editor

1. For a step in an Ixia N2X session, in the **Action** cell, select **command**.

2 Select the command from the drop-down list in the **Description** cell.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Command set

Slot and port numbering starts with 1 (1-based).

Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.

Use the * wildcard character to represent all slots or ports

capture save *port file_URI* [*startPacket*] [*packetCount*]	Saves captured data from the specified port to a file. This command enables you to perform additional analysis using your preferred tool. Defaults: Packet count: 100 Start packet: 1
capture start [*ports*]	Starts capturing on all or specified ports
capture stop [*ports*]	Stops capture on all or specified ports.
collision start [*ports*]	Starts generating collisions on all or specified ports
collision stop [*ports*]	Stops generating collisions on all or specified ports

configuration get [*ports*]	Displays the current configuration settings for all or specified ports. Both **configuration get** and **configuration save** perform replacements on the file so that the specific information (like IP address and port numbers) are translated into variables or command substitutions.
configuration load *fileURI*	Configures the device with information from the specified file. (Typically, you will have created the file previously by executing **configuration save**.) Both **configuration get** and **configuration save** perform replacements on the file so that the specific information (like IP address and port numbers) are translated into variables or command substitutions.
configuration save *fileURI ports*	Saves the current configuration information returned by the device into a specified file. You can use **configuration load** to load the configuration settings to the device.
deselect all stream *ports*	Removes all stream groups from statistics for the specified ports
deselect stream *streamGroup*	Removes specified stream group from statistics for the specified group
eval *TclStatement*	Evaluates the specified Tcl statement
exit	Disconnects iTest from the session and leaves the session running. See **reboot**
help	Displays the list of Ixia N2X session commands
reboot	Disconnects iTest from the session, closes the session, and reboots the modules that are in use. **Note** The ports typically require 3-7 minutes to reboot with the new configuration. For this reason, We recommend that you reuse session labels in test suites so that there is no need to use **reboot**. See **exit**
select all stream [*ports*]	Selects all stream groups for statistics for all or specified ports. Used for stream-based statistics.
select field name *ports field*	For field-based statistics: Selects the specified field name on specified port for which statistics will be collected. See **select field values**
select field values *ports initValue count incr*	Selects values for which to display statistics for the field specified by the **select field name** command.
select stream *streamGroup*	Selects the specified stream group for statistics

show packet *ports packetNum*	Displays details about the specified captured packet on the specified ports **Note** The response to **show packet** differs on each of the supported platforms (Linux, Solaris, and Windows) See **show packet raw**
show packet list *ports startPacket count*	Returns a table that lists several packets (starting with *startPacket* for the number of packets specified by *count*) on the specified port.
show packet raw *ports packetNum*	Displays packet, but without details—just the hex dump. See **show packet**
show stats [*ports*]	Displays a table of statistics by port on all or specified ports
show stats field [*ports*]	Displays field-based statistics on all or specified ports Use the **select field name** command to select a packet field. Now select the desired field values using the **select field values** command. The **show stats field** command displays a table statistics based on the field value. For instance, transmitted bytes with vlan_id = 3, received bytes with vlan_id = 5, and so on.
show stats stream [*ports*]	Displays stream-based statistics on all or specified ports
show streams	Show information about all session-related streams (name, source ports, destination ports, and is enabled are included)
stream enable *streamGroup*	Enables transmission on the specified stream
stream disable *streamGroup*	Disables transmission on the specified stream
transmit start [*ports*]	Starts transmitting on all or specified ports
transmit stop [*ports*]	Stops transmitting on all or specified ports

Session profile property settings for Ixia N2X sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

N2XTraffic

N2X session label	Specify the Session Label to use on the Ixia controller. If a session with that label already exists, it will be used, otherwise, a new session is created.
N2X chassis IP address	Specify the IP address or hostname of the device.
Cards (slots)	When you specify a card for the session, all of its ports are selected by default. Specify a card using a single number. Specify multiple cards using a series of numbers separated by commas. Wildcard characters are not supported. Alternatively, you can specify particular individual ports using the **Ports** property. **Note** You typically supply a value for either the **Cards** property or the **Ports** property. **Examples** To specify all ports on card 2, set the **Cards** property to **2** To specify all ports on cards 2 and 3, set the **Cards** property to **2, 3**
Ports	Specify particular ports to use for the session. (To specify all ports on a card, leave the **Ports** property blank and use the **Cards** property.) Use **<card_a>:<ports>,<card_b>:<ports>** syntax. Wildcard characters are not supported. **Note** You typically supply a value for either the **Cards** property or the **Ports** property. To specify all ports on a card, use the **Cards** property. **Examples** To specify card 2, port 1, use **2:1** To specify multiple cards and ports, use **2:1,3:2,4:3** To specify all ports on card 2, leave the **Ports** property blank and set the **Cards** property to **2**

N2X Traffic > Backward Compatibility

Generate responses for show stats in 3.1.x format	Responses to the show stats command differed in iTest 3.1.x. In iTest 3.1.x, the column titles were stream directions (for example, 101:1 -> 101:2) In iTest 3.2 and later, the column titles are the stream names and the StreamName row has been removed from the response. Checking the box results in3.1.x format responses to show stats commands. Default: Unchecked

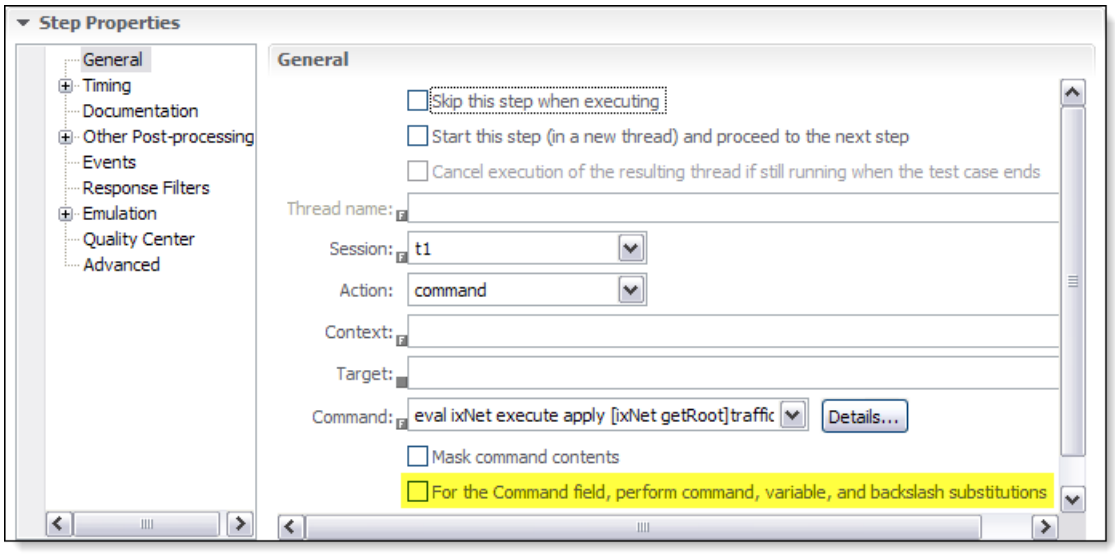
- ### Perform substitution on Ixia commands

  Test case steps that use the iTest **command** action to perform Ixia commands can include proper Ixia syntax that would be incorrectly substituted by the iTest interpreter. Here is an example Ixia command that applies traffic items onto the configuration in Tcl:

  eval ixNet execute apply [ixNet getRoot]traffic::ixNet::OK

By default, the iTest interpreter would read **[ixNet getRoot]** as something to substitute and then fail while trying to execute a (nonexistent) iTest **ixNet** command.

For this reason, be sure to configure each **command** step to ***not*** perform command substitution: In the step properties, uncheck the **For the Command field, perform command, variable, and backslash substitutions** checkbox.



Tcl Interpreter

Ixia sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the session profile, or you can set a preference for all Ixia sessions. You can specify the path to the correct interpreter here, in the session profile. Refer to Ixia documentation for additional Tcl version information.

For example, if Ixia N2X software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the Ixia N2X session. If, instead, you change the preference for all Ixia N2X sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Ixia N2X sessions, use the Preference page. See “Session profile property settings for Ixia N2X sessions” on page 1114.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>

Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Setting preferences for Ixia sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Ixia <type>.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if Ixia Traffic software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the Ixia Traffic session. By changing the preference under **Session Types > Ixia Traffic**, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case

to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: Auto-select
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked
Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

Example Ixia N2X session

```
Spirent Ixia N2X command interpreter. Copyright (c) 2005 - 2012, Spirent
Communications, Inc.


Working platform: Windows

Loading Tcl package  'AgtClient': OK
     Package 'AgtClient' successfully loaded

Loading helper scripts: OK

Searching for QuickScript directory...OK

Connecting to Ixia chassis at 10.155.2.37 ...OK

Checking version: OK

Checking location of 'tethereal.exe' in PATH: OK
     'tethereal.exe' found at 'C:\Program Files\Ethereal'


Connected to session 37 with label Administrator

Ixia N2X>configuration load project://project/myconfig
Configuration loaded from file D:\work\itest\project\myconfig
Ixia N2X>select all stream
Ixia N2X>select field name 101:1,101:2 vlan_priority1
Ixia N2X>select field values 101:1 3 1 0
Ixia N2X>select field values 101:2 1 1 0
Ixia N2X>transmit start
Transmit started
Ixia N2X>capture start
Capture started
Ixia N2X>capture stop
Capture stopped
Ixia N2X>transmit stop
Transmit stop sent. Waiting until test stops... OK.
Ixia N2X>show stats
STATS                       101:1     101:2     101:3   101:4
------------------------------------------------------------------------------
------
AGT_TEST_PACKETS_TRANSMITTED        364826    364826    0     0
AGT_STREAM_PACKETS_TRANSMITTED       0        0        0     0
AGT_TEST_PACKETS_RECEIVED          303086    303086    0     0
AGT_STREAM_PACKETS_RECEIVED         0        0        0     0
AGT_FIELD_TEST_PACKETS_RECEIVED      0        0        0     0
AGT_TEST_OCTETS_TRANSMITTED         24078516   24078516   0     0
AGT_STREAM_OCTETS_TRANSMITTED        0        0        0     0
AGT_TEST_OCTETS_RECEIVED           20003676   20003676   0     0
AGT_STREAM_OCTETS_RECEIVED          0        0        0     0
AGT_STREAM_PACKET_LOSS             0        0        0     0
AGT_PACKET_LATENCY_SUM             493226641   492512049   0     0
AGT_STREAM_LATENCY_SUM             0        0        0     0
AGT_FIELD_LATENCY_SUM              0        0        0     0
AGT_ATM_CELLS_TRANSMITTED           0        0        0     0
AGT_ATM_CELLS_RECEIVED             0        0        0     0
AGT_ETHERNET_OCTETS_TRANSMITTED      24078696   24078696   16372   16372
AGT_ETHERNET_FRAMES_TRANSMITTED      364828    364828    255    255
AGT_ETHERNET_OCTETS_RECEIVED         20005896   20005896   0     0
```

```
AGT_ETHERNET_FRAMES_RECEIVED        303109     303109     0     0
AGT_HDLC_OCTETS_TRANSMITTED_PRESTUFF   0        0        0     0
AGT_HDLC_FRAMES_TRANSMITTED         0        0        0     0
AGT_HDLC_OCTETS_RECEIVED_POST_DESTUFF  0        0        0     0
AGT_HDLC_FRAMES_RECEIVED           0        0        0     0

Ixia N2X>show stats field
PORTS                       101:1           101:2
-------------------------------------------------------------------------------
--
FIELDVALUE                    3              1
AGT_TEST_PACKETS_TRANSMITTED        0              0
AGT_STREAM_PACKETS_TRANSMITTED       0              0
AGT_TEST_PACKETS_RECEIVED          0              0
AGT_STREAM_PACKETS_RECEIVED         0              0
AGT_FIELD_TEST_PACKETS_RECEIVED      303086           303086
AGT_TEST_OCTETS_TRANSMITTED         0              0
AGT_STREAM_OCTETS_TRANSMITTED        0              0
AGT_TEST_OCTETS_RECEIVED           0              0
AGT_STREAM_OCTETS_RECEIVED          0              0
AGT_STREAM_PACKET_LOSS             0              0
AGT_PACKET_LATENCY_SUM             0              0
AGT_STREAM_LATENCY_SUM             0              0
AGT_FIELD_LATENCY_SUM             493226641         492512049
AGT_ATM_CELLS_TRANSMITTED          0              0
AGT_ATM_CELLS_RECEIVED             0              0
AGT_ETHERNET_OCTETS_TRANSMITTED      0              0
AGT_ETHERNET_FRAMES_TRANSMITTED      0              0
AGT_ETHERNET_OCTETS_RECEIVED        0              0
AGT_ETHERNET_FRAMES_RECEIVED        0              0
AGT_HDLC_OCTETS_TRANSMITTED_PRESTUFF   0              0
AGT_HDLC_FRAMES_TRANSMITTED         0              0
AGT_HDLC_OCTETS_RECEIVED_POST_DESTUFF  0              0
AGT_HDLC_FRAMES_RECEIVED           0              0

Ixia N2X>show stats stream
GroupName                    StreamGroup 1  StreamGroup 2
-------------------------------------------------------------------------------
------
StreamIndex                   0          0
AGT_TEST_PACKETS_TRANSMITTED        0          0
AGT_STREAM_PACKETS_TRANSMITTED       88663        88663
AGT_TEST_PACKETS_RECEIVED          0          0
AGT_STREAM_PACKETS_RECEIVED         88663        88663
AGT_FIELD_TEST_PACKETS_RECEIVED      0          0
AGT_TEST_OCTETS_TRANSMITTED         0          0
AGT_STREAM_OCTETS_TRANSMITTED        5851758      5851758
AGT_TEST_OCTETS_RECEIVED           0          0
AGT_STREAM_OCTETS_RECEIVED          5851758      5851758
AGT_STREAM_PACKET_LOSS             0          0
AGT_PACKET_LATENCY_SUM             0          0
AGT_STREAM_LATENCY_SUM             58789208     58952868
AGT_FIELD_LATENCY_SUM             0          0
AGT_ATM_CELLS_TRANSMITTED          0          0
AGT_ATM_CELLS_RECEIVED             0          0
AGT_ETHERNET_OCTETS_TRANSMITTED      0          0
AGT_ETHERNET_FRAMES_TRANSMITTED      0          0
AGT_ETHERNET_OCTETS_RECEIVED        0          0
AGT_ETHERNET_FRAMES_RECEIVED        0          0
AGT_HDLC_OCTETS_TRANSMITTED_PRESTUFF   0          0
AGT_HDLC_FRAMES_TRANSMITTED         0          0
```

```
AGT_HDLC_OCTETS_RECEIVED_POST_DESTUFF  0         0
AGT_HDLC_FRAMES_RECEIVED          0         0

Ixia N2X>show streams
Name       | Source | Destination | Enabled | InStatistics
-------------------------------------------------------------------------------
-----
StreamGroup 1 | 101:1  | 101:1;101:2 | True   | True
StreamGroup 2 | 101:2  | 101:1     | True   | True

Ixia N2X>show packet list 101:1 1 5
Id  Time       Source      Destination    Protocol  Octets            Length
-------------------------------------------------------------------------------
------
1   00:39:51.300072340  00 00 c0 02 01 02   00 00 00 00 00 00   61     00 00 00 00
00 00 00 00 c0 02 ...  66
2   00:39:51.300141940  00 00 c0 02 01 02   00 00 00 00 00 00   61     00 00 00 00
00 00 00 00 c0 02 ...  66
3   00:39:51.300211540  00 00 c0 02 01 02   00 00 00 00 00 00   61     00 00 00 00
00 00 00 00 c0 02 ...  66
4   00:39:51.300281140  00 00 c0 02 01 02   00 00 00 00 00 00   61     00 00 00 00
00 00 00 00 c0 02 ...  66
5   00:39:51.300346390  00 00 c0 02 01 02   00 00 00 00 00 00   61     00 00 00 00
00 00 00 00 c0 02 ...  66

Ixia N2X>show packet 101:1 4
General information
  Number: 1
  Packet Length: 66
  Captured Length: 66
  Captured Time: Sep 13, 2007 13:44:11.000000000
Frame 1 (66 bytes on wire, 66 bytes captured)
  Arrival Time: Sep 13, 2007 13:44:11.000000000
  Time delta from previous packet: 0.000000000 seconds
  Time since reference or first frame: 0.000000000 seconds
  Frame Number: 1
  Packet Length: 66 bytes
  Capture Length: 66 bytes
  Protocols in frame: eth:data
Ethernet II, Src: 00:00:c0:02:01:02 (00:00:c0:02:01:02), Dst:
00:00:00:00:00:00 (00:00:00:00:00:00)
  Destination: 00:00:00:00:00:00 (00:00:00:00:00:00)
    Address: 00:00:00:00:00:00 (00:00:00:00:00:00)
    .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
    .... ..0. .... .... .... .... = Locally Administrated Address: This is a
FACTORY DEFAULT address
  Source: 00:00:c0:02:01:02 (00:00:c0:02:01:02)
    Address: 00:00:c0:02:01:02 (00:00:c0:02:01:02)
    .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
    .... ..0. .... .... .... .... = Locally Administrated Address: This is a
FACTORY DEFAULT address
  Type: Unknown (0x9100)

0000  00 00 00 00 00 00 00 00 C0 02 01 02 91 00 60 00  ........À...‘.`.
0010  81 00 00 00 08 00 45 00 00 28 00 00 00 00 40 3D  _.....E..(....@=
0020  F8 91 C0 02 01 02 C0 01 01 02 00 10 0F FF 00 02  ø‘À...À......ÿ..
0030  6A E2 5C 24 C1 FD 40 CA F1 AE 3F A7 00 00 76 F9  jâ\$Áý@Êñ®?§..vù
0040  C8 B9                              È¹

Ixia N2X>show packet raw 101:1 3
0000  00 00 00 00 00 00 00 00 C0 02 01 02 91 00 60 00  ........À...‘.`.
```

```
0010  81 00 00 00 08 00 45 00 00 28 00 00 00 00 40 3D  _.....E..(....@=
0020  F8 91 C0 02 01 02 C0 01 01 02 00 10 0F FF 00 02  ø‘À...À......ÿ..
0030  6A E1 5C 24 A6 CA FF E8 50 7B 96 8E 00 00 96 D2  já\$¦ÊÿèP{–_..–Ò
0040  37 BE                                           7¾

Ixia N2X>eval AgtInvoke AgtStreamGroup SetExpectedDestinationPorts 7 2
Ixia N2X>show streams
Name       | Source | Destination | Enabled | InStatistics
-------------------------------------------------------------------------------
-----
StreamGroup 1 | 101:1  | 101:2     | True   | True
StreamGroup 2 | 101:2  | 101:1     | True   | True

Ixia N2X>
```

CHAPTER 58

Ixia Traffic sessions

Ixia Traffic session window

The Ixia Traffic session window is an interactive terminal where you enter commands to perform Ixia Traffic actions on the device. The session on the Ixia device returns text responses.

In an interactive Ixia Traffic session, you can:

- Request and display statistics, start and stop traffic, start and stop collisions, and perform other functions commonly used during a test.
- Get the current configuration settings from an Ixia device. The **configuration save** command requests the current device configuration settings and then saves the response to a specified file. The settings are returned in the response as a Tcl script. The **configuration get** command requests the same information and displays it on the terminal.

  Note During interactive Ixia Traffic sessions, both **configuration get** and **configuration save** replace specific information in the configuration file (like IP address and port numbers) with variables so that you can set values dynamically in test case steps.

- Use the **configuration load** command to load the configuration settings (from the file generated by the **configuration save** command) into the Ixia device. When you save an interactive session into a test case, a **configuration load** step is saved as the **configure** Action to send the configuration information to the device.

Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from the Ixia device to perform Layer 2-3 testing.

Auto-generated queries for analysis

iTest auto-maps Ixia device iTest auto-maps IxNetwork responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries.Errors while starting traffic

Saving an Ixia configuration into a iTest test case

Note During Ixia sessions, if the following error codes are reported while attempting to start traffic, there is most likely an issue with the Ixia configuration file.

```
ixNet::ERROR-ErrorsOccurred-6304-Cannot start traffic or
ixNet::ERROR-ErrorsOccurred-6305-Error in applying traffic
```

Work with Ixia support to resolve the issue.

1. In IxExplorer, configure streams in the normal way.
2. Start a iTest Ixia Traffic session and issue the **configuration get** command.
3. The **configure** command in the resulting test case contains the full chassis configuration Tcl file.

   To view the configuration file in the Response view, click the **configure** step. You can copy and paste the step into other test cases.

Note The following errors when starting an Ixia session are typically caused by issues with the configuration file:

```
ixNet::ERROR-ErrorsOccurred-6304-Cannot start traffic
ixNet::ERROR-ErrorsOccurred-6305-Error in applying traffic
```

Ixia Traffic command set

This topic describes all Ixia Traffic commands that you can perform manually in an interactive Ixia Traffic session and that a iTest test case can execute.

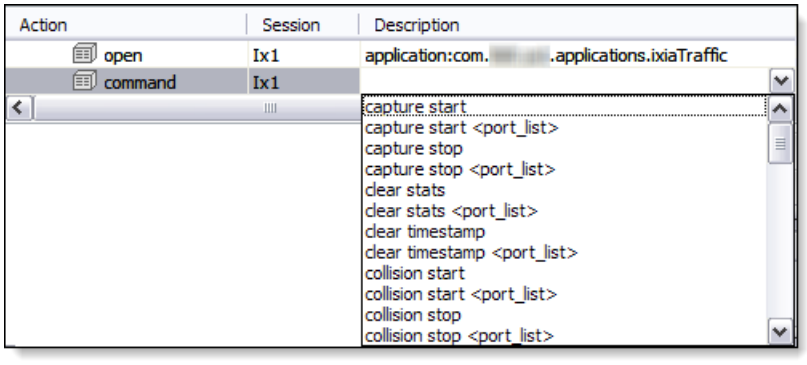
iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or apply analysis rules to the values of interest in the response.

While working in an interactive session

- To view the list of commands while working in an interactive Ixia Traffic session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **stream send** command, type **stream send ?**.

While working on a test case in the Test Case editor

1. For a step in an Ixia Traffic session, in the **Action** cell, select **command**.
2. Select the command from the drop-down list in the **Description** cell.



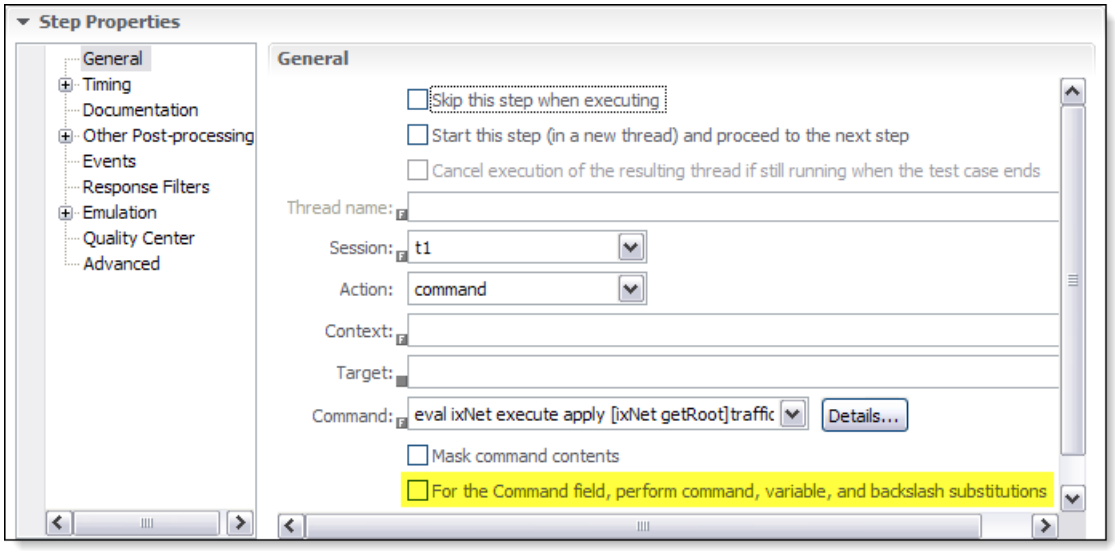
Perform substitution on Ixia commands

Test case steps that use the iTest **command** action to perform Ixia commands can include proper Ixia syntax that would be incorrectly substituted by the iTest interpreter. Here is an example Ixia command that applies traffic items onto the configuration in Tcl:

eval ixNet execute apply [ixNet getRoot]traffic::ixNet::OK

By default, the iTest interpreter would read **[ixNet getRoot]** as something to substitute and then fail while trying to execute a (nonexistent) iTest **ixNet** command.

For this reason, be sure to configure each **command** step to ***not*** perform command substitution: In the step properties, uncheck the **For the Command field, perform command, variable, and backslash substitutions** checkbox.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Command reference

You can use the following commands when defining steps in a test case.

- Slot and port numbering starts with zero (zero-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.
- Use the * wildcard character to represent all slots or ports

capture start [*portList*]	Start capturing on all or specified ports
capture stop [*portList*]	Stop capture on all or specified ports.

clear stats [*portList*]	Clear statistics for all or specified ports
clear timestamp [*portList*]	Clear the timestamp being used in generated packets on all or specified ports
collision start [*portList*]	Start generating collisions on all or specified ports
collision stop [*portList*]	Stop generating collisions on all or specified ports
configuration get [**mode** {**full | comment | nondefault**}] [portList]	Display the current Ixia configuration settings for all or specified ports. During interactive Ixia Traffic sessions, both **configuration get** and **configuration save** replace specific information in the configuration file (like IP address and port numbers) with variables so that you can set values dynamically in test case steps. The optional **mode** argument specifies how the configuration file will be generated. If mode is not specified, then **nondefault** is used. **full** - Generate configuration file with all information (including defaults) **comment** - Generate configuration file with all information (including commented defaults) **nondefault** – (default if **mode** is not specified) Generate configuration file without default information
configuration load *URI*	Configure the Ixia device with information from the specified file. (Typically, you will have created the file previously by executing **configuration save**.) When you save an interactive session into a test case, a **configuration load** step is saved as a step with an Action of **configure** (to send the configuration file to the Ixia Traffic session). **Note** The **configuration save** and **configuration load** commands use only the first port specified in the session profile **Port** property. For example, if you specify a **Port** property of **1:3,1:2** in the session profile, then **configuration load** loads 1:3 and ignores any port specified in a preceding **configuration save** step. **Workaround**: Use **configuration save** and **configuration load** commands with no arguments. To modify ports, specify them in the session profile. **Note** The following errors when starting an Ixia session are typically caused by issues with the configuration file: ixNet::ERROR-ErrorsOccurred-6304-Cannot start traffic ixNet::ERROR-ErrorsOccurred-6305-Error in applying traffic

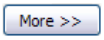
configuration save URI [portList] **[mode full | comment | nondefault]**	Save the current configuration information configuration returned by ScriptGen into a specified file for all or specified ports. You can use configuration load to load the configuration settings to the Ixia device. You can specify the resulting script file in a session profile to configure the device. As a result, when the session starts, the device is configured exactly as if you had configured it using IxExplorer. During interactive Ixia Traffic sessions, both **configuration get** and **configuration save** replace specific information in the configuration file (like IP address and port numbers) with variables so that you can set values dynamically in test case steps. The optional **mode** argument specifies how the configuration file will be generated. If mode is not specified, then **nondefault** is used. **full** - Generate configuration file with all information (including defaults) **comment** - Generate configuration file with all information (including commented defaults) **nondefault** – (default if **mode** is not specified) Generate configuration file without default information. **Note** The **configuration save** and **configuration load** commands use only the first port specified in the session profile **Port** property. For example, if you specify a **Port** property of **1:3,1:2** in the session profile, then **configuration load** loads 1:3 and ignores any port specified in a preceding **configuration save** step. **Workaround**: Use **configuration save** and **configuration load** commands with no arguments. To modify ports, specify them in the session profile.
eval *TclStatement*	Evaluate a Tcl statement for direct access to the IxTclHal API
exit	Unlock ports, disconnect from the device, and then close the Ixia Traffic session.
help	Display all command syntax and descriptions
show capture *startPacket num_of_packets* [*port*]	Display summary information about captured packets for all or specified ports
show packet *port packet*	Display details about a captured packet
show packet *port packet* **raw**	Display details about a captured packet in raw format
show rates [*portList*]	Displays a table of packet rates for all ports or by port
show stats [*portList*]	Displays a table of statistics for all ports or by port
show streams [*portList*]	Displays a table of statistics for streams on all or specified ports.
stream send *port streamID*	Transmit a specified stream on a specified port
stream enable *port streamID*	Enable transmission of a specified stream on a specified port
stream disable *port streamID*	Disable transmission of a specified stream on a specified port
transmit pause [*portList*]	Pause transmission on all or specified ports
transmit stagger [*portList*]	Start staggered transmit on all or specified ports

transmit start [*portList*]	Start transmitting on all or specified ports
transmit step [*portList*]	Send one round of traffic on all or specified ports
transmit stop [*portList*]	Stop transmitting on all or specified ports

Session profile property settings for Ixia Traffic sessions

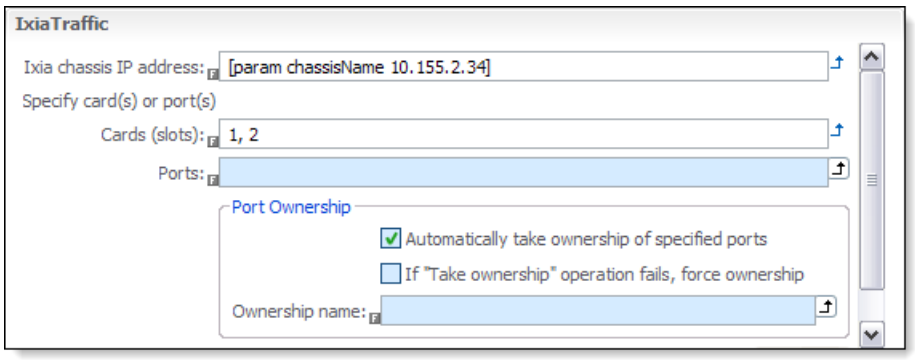
This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Ixia Traffic



Ixia chassis IP address	Specify the IP address or hostname of the device. In the example, the **Ixia chassis IP address** property value is specified dynamically at runtime by using a param command to pass the **chassisName** parameter.
IxOS location	Some tests are designed to run once using a particular IxOS version and then to run again using a different IxOS version (for example, a run using 5.60 followed by a run using 5.70). Remember that each session profile is associated with a single IxOS version.To enable a test to run on multiple IxOS versions, you must first create a session profile for each IxOS version — the test must then use each session profile in turn. Use the **IxOS location** property to specify the IxOS version for this particular session profile. **To set the default path for IxOS** **Windows** — Add the path to the **Path** environment variable. If you do not specify a default location, iTest uses the latest IxOS in the registry. **Linux** — Add the path to **IXIA_HOME**. If you do not specify a default location, then no default is set. **To include an IxOS version in the drop-down list** Specify the version in the IXIA_MULTI_VERSIONS path variable. **Note** If the IxOS versions are currently listed in the registry key, then you do not have to create the IXIA_MULTI_VERSIONS path variable. **Example** There are two IxOS versions installed on the computer: • C:\Program Files\Ixia\IxOS\6.10-EA-SP2 • C:\Program Files\Ixia\IxOS\6.20-EA-SP1 Create the IXIA_MULTI_VERSIONS path variable and set it to: C:\Program Files\Ixia\IxOS\6.10-EA-SP2;C:\Program Files\Ixia\IxOS\6.20-EA-SP1

Cards (slots)	When you specify a card for the session, all of its ports are selected by default. Specify a slot using a single number. Specify multiple slots using a series of numbers separated by commas. Alternatively, you can specify particular individual ports using the **Ports** property. **Note**: You typically supply a value for either the **Cards** property or the **Ports** property.
Ports	Specify the ports to use for the session. Use **<card_a>:<ports>,<card_b>:<ports>** syntax. You can use the ***** wildcard character to indicate all cards. **Note** To specify all ports on a card, use the **Cards** property.You typically supply a value for either the **Cards** property or the **Ports** property. **Examples** To specify card 2, port 1, use **2:1** To specify multiple cards and ports, use **2:1,3:2,4:3** **Note** iTest uses the ports in the order specified in the session profile. For example, If the session profile specifies **1:2,1:3**, and you save the Ixia config file using only one port, then configuration load command will always pick **1:2** as the port for loading. If you want to load **1:3**, then you need either to change the order of the ports in the session profile, or to create a new session profile that specifies just **1:3**.
Automatically take ownership	Upon connecting, take ownership of the chassis so no other user can submit commands.
If "Take ownership" operation fails	Upon connecting, take ownership of the chassis even if another user currently has ownership.
Ownership name	Specify a name to associate with your ownership that other users will see.
Configuration script	Optional: Specify the script that was generated using the Ixia Traffic **configuration save** command. As a result, when the session starts, the device is configured exactly as if you had configured it using IxExplorer. **Note** The following errors when starting an Ixia session are typically caused by issues with the configuration file: ixNet::ERROR-ErrorsOccurred-6304-Cannot start traffic ixNet::ERROR-ErrorsOccurred-6305-Error in applying traffic See "Ixia Traffic command set" on page 1127 and "Ixia Traffic session window" on page 1126

Tcl Interpreter

Ixia sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the session profile, or you can set a preference for all Ixia sessions. You can specify the path to the correct interpreter here, in the session profile. Refer to Ixia documentation for additional Tcl version information.

For example, if Ixia N2X software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the Ixia N2X session. If, instead, you change the preference for all Ixia N2X sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Ixia N2X sessions, use the Preference page. See “Session profile property settings for Ixia N2X sessions” on page 1114.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**

Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Example Ixia Traffic session

```
Spirent Ixia Traffic command interpreter. Copyright (c) 2005 - 2008, Spirent
Communications, Inc.

Working platform: Windows

Checking location of 'ixTclHal.dll' in PATH: OK
     'ixTclHal.dll' found at 'C:\Program Files\Ixia'

Loading Tcl package  'IxTclHal': OK
     Package 'IxTclHal' successfully loaded

Loading Tcl package  'Scriptgen': OK
     Package 'Scriptgen' successfully loaded

Checking location of 'tethereal.exe' in PATH: OK
     'tethereal.exe' found at 'C:\Program Files\Ethereal'

Connecting to Ixia chassis at 10.155.2.23: OK

Ixia Traffic>stream disable 1:1 0
Ixia Traffic>transmit start
Ixia Traffic>transmit stop
Ixia Traffic>show stats
STATS                  1:1:1      1:1:2      1:1:3      1:1:4
-------------------------------------------------------------------------------
---------------------
alignmentErrors            0          0          0          0
```

```
bitsReceived              573075456    2748907008   1198046208   1198046208
bitsSent                 2748907008   573075456    1198046208   1198046208
bytesReceived              71634432    343613376    149755776    149755776
bytesSent                 343613376    71634432    149755776    149755776
captureFilter               1119288     5368959     2339934     2339934
captureState               0         0         0         0
captureTrigger              1119288     5368959     2339934     2339934
collisionFrames            0         0         0         0
collisions                0         0         0         0
dribbleErrors              0         0         0         0
duplexMode                1         1         1         1
enableArpStats              1         1         1         1
enableAtmOamStats           0         0         0         0
enableBgpStats             -         -         -         -
enableDhcpStats             0         0         0         0
```

```
enableDhcpV6Stats           0         0         0         0
enableEigrpStats            -         -         -         -
enableIcmpStats             1         1         1         1
enableIgmpStats             -         -         -         -
enableIsisStats             -         -         -         -
enableLdpStats              -         -         -         -
enableMldStats              -         -         -         -
enableOspfStats             -         -         -         -
enableOspfV3Stats           -         -         -         -
enablePimsmStats            -         -         -         -
enablePosExtendedStats      1         1         1         1
enableProtocolServerStats   1         1         1         1
enableRsvpStats             -         -         -         -
enableStpStats              -         -         -         -
enableTemperatureSensorsStats  1      1         1         1
enableUsbExtendedStats      -         -         -         -
excessiveCollisionFrames    0         0         0         0
fcsErrors                   0         0         0         0
flowControlFrames           0         0         0         0
fragments                   0         0         0         0
framesReceived              1119288   5368959   2339934   2339934
framesSent                  5368959   1119288   2339934   2339934
includeRprPayloadFcsInCrc   1         1         1         1
lateCollisions              0         0         0         0
lineSpeed                   100       100       100       100
link                        1         1         1         1
mode                        0         0         0         0
numCapturePackets           0         0         0         0
oversize                    0         0         0         0
pauseState                  0         0         0         0
protocolServerVlanDroppedFrames  0    0         0         0
rxArpReply                  0         0         0         0
rxArpRequest                0         0         0         0
rxPingReply                 0         0         0         0
rxPingRequest               0         0         0         0
scheduledTransmitTime       0         0         0         0
symbolErrors                0         0         0         0
transmitDuration            0         7521614520  7521614520  7521614520
transmitState               0         0         0         0
txArpReply                  0         0         0         0
txArpRequest                0         0         0         0
txPingReply                 0         0         0         0
txPingRequest               0         0         0         0
undersize                   0         0         0         0
userDefinedStat1            0         0         0         0
userDefinedStat2            0         0         0         0
vlanTaggedFramesRx          0         0         0         0
Ixia Traffic>
Ixia Traffic>stream enable 1:1 0
Ixia Traffic>transmit start
Ixia Traffic>transmit stop
Ixia Traffic>show stats
STATS                  1:1:1      1:1:2      1:1:3      1:1:4
-------------------------------------------------------------------------------
--------------------
alignmentErrors             0         0         0         0
bitsReceived                991925760    3167757312   1616896512   1616896512
bitsSent                    3167757312   991925760    1616896512   1616896512
bytesReceived               123990720    395969664    202112064    202112064
bytesSent                   395969664    123990720    202112064    202112064
captureFilter               1937355     6187026     3158001     3158001
```

```
captureState              0         0         0         0
captureTrigger            1937355     6187026     3158001     3158001
collisionFrames           0         0         0         0
collisions                0         0         0         0
dribbleErrors             0         0         0         0
duplexMode                1         1         1         1
enableArpStats            1         1         1         1
enableAtmOamStats         0         0         0         0
enableBgpStats            -         -         -         -
enableDhcpStats           0         0         0         0
enableDhcpV6Stats         0         0         0         0
enableEigrpStats          -         -         -         -
enableIcmpStats           1         1         1         1
enableIgmpStats           -         -         -         -
enableIsisStats           -         -         -         -
enableLdpStats            -         -         -         -
enableMldStats            -         -         -         -
enableOspfStats           -         -         -         -
enableOspfV3Stats         -         -         -         -
enablePimsmStats          -         -         -         -
enablePosExtendedStats       1         1         1         1
enableProtocolServerStats     1         1         1         1
enableRsvpStats           -         -         -         -
enableStpStats            -         -         -         -
enableTemperatureSensorsStats    1         1         1         1
enableUsbExtendedStats       -         -         -         -
excessiveCollisionFrames      0         0         0         0
fcsErrors                 0         0         0         0
flowControlFrames         0         0         0         0
fragments                 0         0         0         0
framesReceived            1937355     6187026     3158001     3158001
framesSent                6187026     1937355     3158001     3158001
includeRprPayloadFcsInCrc     1         1         1         1
lateCollisions            0         0         0         0
lineSpeed                 100       100       100       100
link                   1         1         1         1
mode                   0         0         0         0
numCapturePackets         0         0         0         0
oversize                  0         0         0         0
pauseState                0         0         0         0
protocolServerVlanDroppedFrames   0         0         0         0
rxArpReply                0         0         0         0
rxArpRequest              0         0         0         0
rxPingReply               0         0         0         0
rxPingRequest             0         0         0         0
scheduledTransmitTime        0         0         0         0
symbolErrors              0         0         0         0
transmitDuration          5497409400   5497409400   5497409400   5497409400
transmitState             0         0         0         0
txArpReply                0         0         0         0
txArpRequest              0         0         0         0
txPingReply               0         0         0         0
txPingRequest             0         0         0         0
undersize                 0         0         0         0
userDefinedStat1          0         0         0         0
userDefinedStat2          0         0         0         0
vlanTaggedFramesRx        0         0         0         0
Ixia Traffic>
Ixia Traffic>capture start
Ixia Traffic>transmit start
Ixia Traffic>transmit stop
```

```
Ixia Traffic>capture stop
Ixia Traffic>show stats
STATS                    1:1:1      1:1:2      1:1:3      1:1:4
-------------------------------------------------------------------------------
--------------------
alignmentErrors             0          0          0          0
bitsReceived             1587878400   3763709952   2212849152   2212849152
bitsSent                 3763709952   1587878400   2212849152   2212849152
bytesReceived             198484800    470463744    276606144    276606144
bytesSent                 470463744    198484800    276606144    276606144
captureFilter              3101325     7350996     4321971     4321971
captureState                0          0          0          0
captureTrigger             3101325     7350996     4321971     4321971
collisionFrames             0          0          0          0
collisions                 0          0          0          0
dribbleErrors               0          0          0          0
duplexMode                 1          1          1          1
enableArpStats              1          1          1          1
enableAtmOamStats           0          0          0          0
enableBgpStats              -          -          -          -
enableDhcpStats             0          0          0          0
enableDhcpV6Stats           0          0          0          0
enableEigrpStats            -          -          -          -
enableIcmpStats             1          1          1          1
enableIgmpStats             -          -          -          -
enableIsisStats             -          -          -          -
enableLdpStats              -          -          -          -
enableMldStats              -          -          -          -
enableOspfStats             -          -          -          -
enableOspfV3Stats           -          -          -          -
enablePimsmStats            -          -          -          -
enablePosExtendedStats         1          1          1          1
enableProtocolServerStats      1          1          1          1
enableRsvpStats             -          -          -          -
enableStpStats              -          -          -          -
enableTemperatureSensorsStats    1          1          1          1
enableUsbExtendedStats         -          -          -          -
excessiveCollisionFrames       0          0          0          0
fcsErrors                  0          0          0          0
flowControlFrames           0          0          0          0
fragments                  0          0          0          0
framesReceived              3101325     7350996     4321971     4321971
framesSent                 7350996     3101325     4321971     4321971
includeRprPayloadFcsInCrc      1          1          1          1
lateCollisions              0          0          0          0
lineSpeed                  100        100        100        100
link                     1          1          1          1
mode                     0          0          0          0
numCapturePackets            18668      18668      18668      18668
oversize                 0          0          0          0
pauseState                0          0          0          0
protocolServerVlanDroppedFrames   0          0          0          0
rxArpReply                0          0          0          0
rxArpRequest               0          0          0          0
rxPingReply               0          0          0          0
rxPingRequest              0          0          0          0
scheduledTransmitTime         0          0          0          0
symbolErrors              0          0          0          0
transmitDuration           7821877560   7821877560   7821877560   7821877560
transmitState              0          0          0          0
txArpReply                0          0          0          0
```

```
txArpRequest              0         0         0         0
txPingReply               0         0         0         0
txPingRequest             0         0         0         0
undersize                 0         0         0         0
userDefinedStat1          0         0         0         0
userDefinedStat2          0         0         0         0
vlanTaggedFramesRx        0         0         0         0
Ixia Traffic>
Ixia Traffic>show packet 1:1 1
Frame 1 (64 bytes on wire, 64 bytes captured)
   Arrival Time: Jan  1, 1970 06:00:00.000000000
   Time delta from previous packet: 0.000000000 seconds
   Time since reference or first frame: 0.000000000 seconds
   Frame Number: 1
   Packet Length: 64 bytes
   Capture Length: 64 bytes
   Protocols in frame: eth:llc
IEEE 802.3 Ethernet
   Destination: 00:01:00:00:00:00 (00:01:00:00:00:00)
      Address: 00:01:00:00:00:00 (00:01:00:00:00:00)
      .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
      .... ..0. .... .... .... .... = Locally Administrated Address: This is a
FACTORY DEFAULT address
   Source: 00:01:00:00:01:00 (00:01:00:00:01:00)
      Address: 00:01:00:00:01:00 (00:01:00:00:01:00)
      .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
      .... ..0. .... .... .... .... = Locally Administrated Address: This is a
FACTORY DEFAULT address
   Length: 1
   Trailer: 030405060708090A0B0C0D0E0F101112131415161718191A...
   Frame check sequence: 0xadb71f60 [correct]
Logical-Link Control
   DSAP: LLC Sub-Layer Management (0x02)
   IG Bit: Individual
[Malformed Packet: LLC]

0000  00 01 00 00 00 00 00 01 00 00 01 00 00 01 02 03  ................
0010  04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13  ................
0020  14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23  ............ !"#
0030  24 25 26 27 28 29 2a 2b 2c 2d 2e 2f ad b7 1f 60  $%&'()*+,-./...`
Ixia Traffic>

Ixia Traffic>show streams
Port                | 4:2           | 4:2
--------------------------------------------------------------------------------
--
--------------------------------------------------------------------------------
--
Stream ID             | 1             | 2
name                |               |
enable                | 1             | 1
enableSuspend           | 0             | 0
region                | 0             | 0
numBursts             | 1             | 1
numFrames             | 100           | 100
ifg                 | 960.0         | 960.0
ifgType               | 0             | 0
ifgMIN                | 1920.0          | 1920.0
ifgMAX                | 2560.0          | 2560.0
ibg                 | 960.0         | 960.0
enableIbg             | 0             | 0
```

```
isg                | 960.0          | 960.0
enableIsg            | 0            | 0
gapUnit              | 0            | 0
percentPacketRate        | 100.0          | 100.0
fpsRate              | 148809.52381     | 148809.52381
bpsRate              | 76190476.1905     | 76190476.1905
rateMode             | 1            | 1
preambleSize           | 8            | 8
preambleData           | 55 55 55 55 55 55 D5 | 55 55 55 55 55 55 D5
sa                 | 00 00 03 00 01 00   | 00 00 03 00 01 00
saRepeatCounter          | 4            | 4
saStep               | 1            | 1
saMaskValue            | 00 00 00 00 00 00   | 00 00 00 00 00 00
saMaskSelect           | 00 00 00 00 00 00   | 00 00 00 00 00 00
enableSaContinueFromLastValue | 0            | 0
da                 | 00 00 03 00 00 00   | 00 00 03 00 00 00
daRepeatCounter          | 4            | 4
daStep               | 1            | 1
daMaskValue            | 00 00 00 00 00 00   | 00 00 00 00 00 00
daMaskSelect           | 00 00 00 00 00 00   | 00 00 00 00 00 00
enableDaContinueFromLastValue | 0            | 0
framesize            | 64           | 64
frameSizeType          | 0            | 0
frameSizeMIN           | 64           | 64
frameSizeMAX           | 64           | 64
frameSizeStep          | 1            | 1
enableTimestamp          | 0            | 0
fcs                | 0            | 0
patternType            | 0            | 0
dataPattern            | 12           | 12
pattern              | 00 01 02 03      | 00 01 02 03
frameType            | FF FF          | FF FF
numDA                | 16           | 16
numSA                | 16           | 16
dma                | 0            | 0
rxTriggerEnable          | 0            | 0
```

Specify how long to wait for the prompt to appear

```
IntEnable          | 1            | 1
loopCount            | 1            | 1
returnToId           | 1            | 1
enforceMinGap         | 12           | 12
enableStatistic       | 1            | 1
enableSourceInterface     | 0            | 0
sourceInterfaceDescription  |              |
startTxDelayUnit       | 4            | 4
startTxDelay         | 0.0          | 0.0
priorityGroup         | 0            | 0
fir                | -1           | -1
suspendState          | 0            | 0
framerate           | 148810         | 148810
warnings            |              |
floatRate           | 148809.52381     | 148809.52381

IxiaTraffic>show streams 4:1
IxiaTraffic>show streams 4:2
Port               | 4:2          | 4:2
-------------------------------------------------------------------------------
--
-------------------------------------------------------------------------------
--
Stream ID           | 1            | 2
name              |            |
enable             | 1            | 1
enableSuspend         | 0            | 0
region             | 0            | 0
numBursts           | 1            | 1
numFrames           | 100          | 100
ifg                | 960.0          | 960.0
ifgType             | 0            | 0
ifgMIN             | 1920.0         | 1920.0
ifgMAX             | 2560.0         | 2560.0
ibg                | 960.0          | 960.0
enableIbg           | 0            | 0
isg                | 960.0          | 960.0
enableIsg           | 0            | 0
gapUnit             | 0            | 0
percentPacketRate       | 100.0          | 100.0
fpsRate             | 148809.52381     | 148809.52381
bpsRate             | 76190476.1905     | 76190476.1905
rateMode            | 1            | 1
preambleSize          | 8            | 8
preambleData          | 55 55 55 55 55 55 D5 | 55 55 55 55 55 55 D5
sa                 | 00 00 03 00 01 00   | 00 00 03 00 01 00
saRepeatCounter         | 4            | 4
saStep             | 1            | 1
saMaskValue           | 00 00 00 00 00 00   | 00 00 00 00 00 00
saMaskSelect          | 00 00 00 00 00 00   | 00 00 00 00 00 00
enableSaContinueFromLastValue | 0            | 0
da                 | 00 00 03 00 00 00   | 00 00 03 00 00 00
daRepeatCounter         | 4            | 4
daStep             | 1            | 1
daMaskValue           | 00 00 00 00 00 00   | 00 00 00 00 00 00
daMaskSelect          | 00 00 00 00 00 00   | 00 00 00 00 00 00
enableDaContinueFromLastValue | 0            | 0
framesize           | 64           | 64
frameSizeType         | 0            | 0
frameSizeMIN          | 64           | 64
```

```
frameSizeMAX           | 64            | 64
frameSizeStep           | 1            | 1
enableTimestamp          | 0           | 0
fcs                   | 0             | 0
patternType             | 0           | 0
dataPattern             | 12          | 12
pattern                | 00 01 02 03     | 00 01 02 03
frameType               | FF FF         | FF FF
numDA                  | 16            | 16
numSA                  | 16            | 16
dma                   | 0             | 0
rxTriggerEnable          | 0           | 0
```

Specify how long to wait for the prompt to appear

```
IntEnable          | 1            | 1
loopCount            | 1            | 1
returnToId           | 1            | 1
enforceMinGap         | 12           | 12
enableStatistic       | 1            | 1
enableSourceInterface     | 0            | 0
sourceInterfaceDescription  |              |
startTxDelayUnit       | 4            | 4
startTxDelay         | 0.0          | 0.0
priorityGroup         | 0            | 0
fir                | -1           | -1
suspendState          | 0            | 0
framerate           | 148810         | 148810
warnings            |              |
floatRate           | 148809.52381     | 148809.52381

IxiaTraffic>show streams 4:2, 4:3
Port               | 4:2          | 4:2
-------------------------------------------------------------------------------
--
-------------------------------------------------------------------------------
--
Stream ID           | 1            | 2
name              |              |
enable             | 1            | 1
enableSuspend         | 0            | 0
region             | 0            | 0
numBursts           | 1            | 1
numFrames           | 100          | 100
ifg                | 960.0        | 960.0
ifgType             | 0            | 0
ifgMIN             | 1920.0         | 1920.0
ifgMAX             | 2560.0         | 2560.0
ibg                | 960.0        | 960.0
enableIbg           | 0            | 0
isg                | 960.0        | 960.0
enableIsg           | 0            | 0
gapUnit             | 0            | 0
percentPacketRate       | 100.0          | 100.0
fpsRate             | 148809.52381     | 148809.52381
bpsRate             | 76190476.1905    | 76190476.1905
rateMode            | 1            | 1
preambleSize          | 8            | 8
preambleData          | 55 55 55 55 55 55 D5 | 55 55 55 55 55 55 D5
sa                 | 00 00 03 00 01 00  | 00 00 03 00 01 00
saRepeatCounter        | 4            | 4
saStep             | 1            | 1
saMaskValue          | 00 00 00 00 00 00  | 00 00 00 00 00 00
saMaskSelect          | 00 00 00 00 00 00  | 00 00 00 00 00 00
enableSaContinueFromLastValue | 0            | 0
da                 | 00 00 03 00 00 00  | 00 00 03 00 00 00
daRepeatCounter        | 4            | 4
daStep             | 1            | 1
daMaskValue          | 00 00 00 00 00 00  | 00 00 00 00 00 00
daMaskSelect          | 00 00 00 00 00 00  | 00 00 00 00 00 00
enableDaContinueFromLastValue | 0            | 0
framesize           | 64           | 64
frameSizeType         | 0            | 0
frameSizeMIN          | 64           | 64
frameSizeMAX          | 64           | 64
```

```
frameSizeStep          | 1            | 1
enableTimestamp         | 0            | 0
fcs                   | 0            | 0
patternType            | 0            | 0
dataPattern            | 12           | 12
pattern               | 00 01 02 03      | 00 01 02 03
frameType              | FF FF          | FF FF
numDA                 | 16           | 16
numSA                 | 16           | 16
dma                   | 0            | 0
rxTriggerEnable         | 0            | 0
```

Specify how long to wait for the prompt to appear

```
IntEnable          | 1            | 1
loopCount             | 1            | 1
returnToId            | 1            | 1
enforceMinGap          | 12           | 12
enableStatistic         | 1            | 1
enableSourceInterface      | 0            | 0
sourceInterfaceDescription   |              |
startTxDelayUnit         | 4            | 4
startTxDelay           | 0.0          | 0.0
priorityGroup          | 0            | 0
fir                   | -1           | -1
suspendState           | 0            | 0
framerate             | 148810         | 148810
warnings              |              |
floatRate             | 148809.52381     | 148809.52381
Ixia Traffic>

Ixia Traffic>show streams count 4:1
0
Ixia Traffic>show streams count 4:2
5
Ixia Traffic>show streams stats 4:2
Stream ID    | 1         | 2         | 3         | 4         | 5
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------
numGroups    | 1         | 1         | 1         | 1         | 1
framesSent   | 0         | 0         | 0         | 0         | 0
frameRate    | 0         | 0         | 0         | 0         | 0
readTimeStamp | 13873543063740 | 13873543063740 | 13873543063740 |
13873543063740 | 13873543063740
lastTimeStamp | 0         | 0         | 0         | 0         | 0

Ixia Traffic>show streams stats 4:2 1, 3, 5
Stream ID    | 1         | 3         | 5
------------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
-----------------------------
numGroups   | 1         | 1         | 1
framesSent   | 0         | 0         | 0
frameRate   | 0         | 0         | 0
readTimeStamp | 13888020027020 | 13888020027020 | 13888020027020
lastTimeStamp | 0         | 0         | 0

Ixia Traffic>show streams stats 4:2 1 2
Stream ID   | 1         | 2
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
-----------------------------
numGroups   | 1         | 1
framesSent   | 0         | 0
frameRate   | 0         | 0
readTimeStamp | 13896281645020 | 13896281645020
lastTimeStamp | 0         | 0

Ixia Traffic>show streams stats 4:2 1
Stream ID   | 1
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
-----------------------------
numGroups   | 1
framesSent   | 0
frameRate   | 0
readTimeStamp | 13921025080940
lastTimeStamp | 0

Ixia Traffic>transmit start
Ixia Traffic>show streams stats 4:2 1
Stream ID   | 1
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
-----------------------------
numGroups   | 1
framesSent   | 1048432
frameRate   | 148802
readTimeStamp | 13934409102680
lastTimeStamp | 13934605415420

Ixia Traffic>transmit stop
Transmit stop sent. Waiting until test stops... OK.

Ixia Traffic>
```

Setting preferences for Ixia sessions

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Session Types > Ixia <type>**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular

session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if Ixia Traffic software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the Ixia Traffic session. By changing the preference under **Session Types > Ixia Traffic**, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	**Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: Auto-select
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked
Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

CHAPTER 46

Mail (SMTP) Sessions

Sending email messages during test execution

You can add steps that construct and send email messages as plain text or HTML text with embedded images during execution. A test case can construct and send as many email messages as are needed. The message body can contain both fixed text and test response and result data (detailed instructions follow).

- You can append content to the body of the message over the course of as many steps as needed.
- You can use field replacements to place response content and parameter values into the subject or message.
- When building multiple separate email messages, use the Session ID value to associate Mail steps with each other as needed. Session IDs for Mail steps are not associated with device Session IDs.
- Mail steps do not generate responses.
- Mail steps that generate errors cause test case failure.

Preparing to send email: Configuring the session profile

Before you use Mail steps in a test case, it's easiest to specify the SMTP server and user credentials in a Mail session profile. You then refer to the session profile in the Mail **open** step.

Constructing and sending a single email message

The easiest way to add an email to a test case is to save a captured Mail session as a procedure in the test case. You can then edit the Mail session steps as described in the following sections (for example, to insert test data or attach files). See “Defining a Mail (SMTP) session” on page 910.

Here’s an example Mail procedure that results from saving a captured Mail session as a procedure:

Here are the Mail session steps. Each part of the message is a separate Mail action.

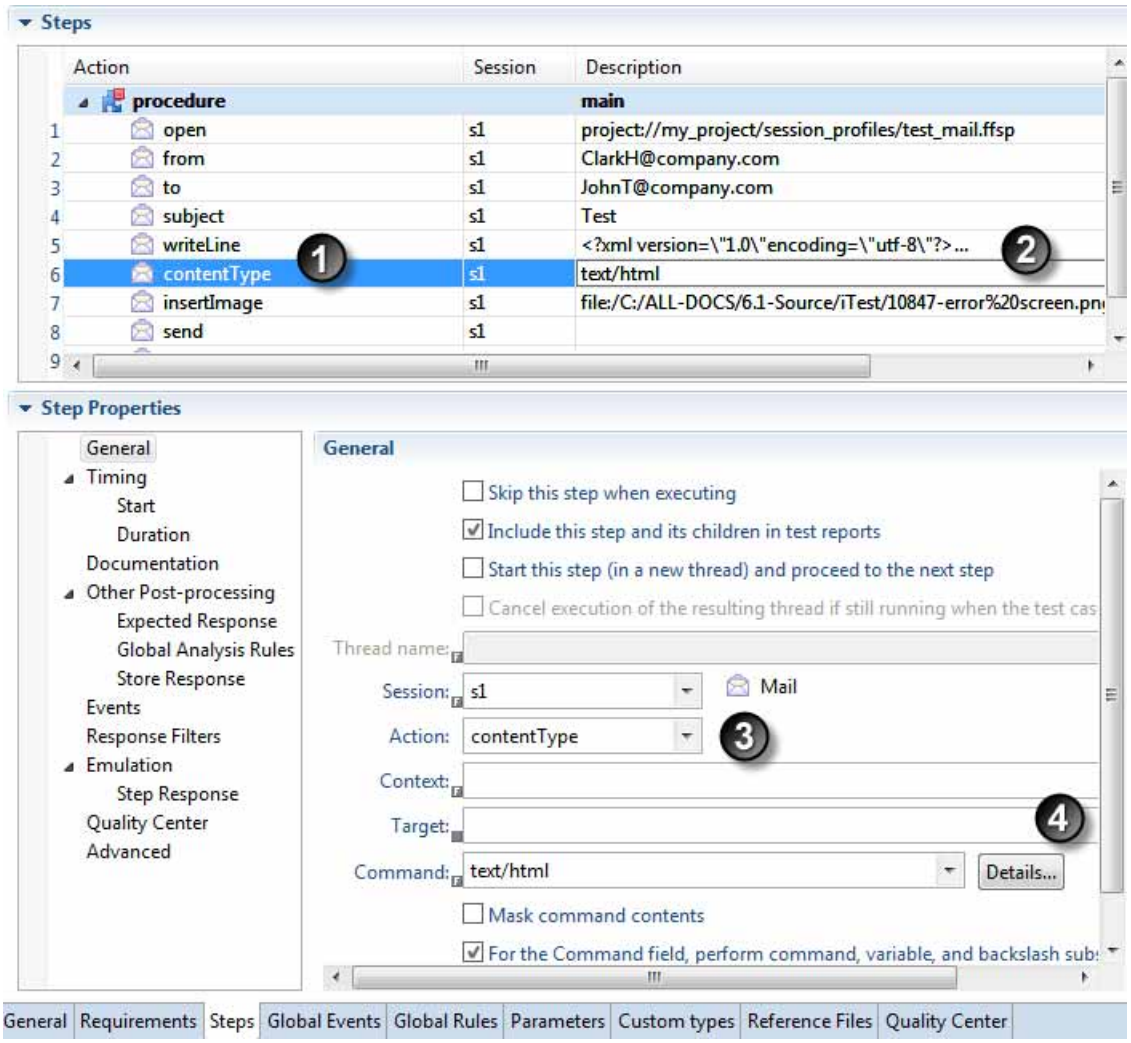
To add response data collected during execution to the message, you'll add **write**, **writeline**, and **message** actions.

The **send** step sends the message.

As with any test case step, you can modify the properties as needed.

For example, you would add field replacements here to include response data in the message body.

1 Click **Advanced** for multi-line contents.



Adding Mail (SMTP) steps manually

1 Add a step and select the **open** action. In the **Description** cell, select **application:com.fnfr.svt.applications.mail**. Session IDs for Mail steps do not interact with Session IDs of other sessions in the test case.

> **Tip** Add new steps by pressing Ctrl+Enter. iTest adds Mail steps in the following order: **from**, **to**, **subject**, **writeLine**, **send**.

2 In the **Action** cell, select a Mail action and, if needed, add information to the **Description** cell. You must supply values for the **to** and **from** email addresses. Where indicated, Field replacements are supported. See Chapter 28, “Field Replacements”.

Mail action	Description If applicable, text required in the Description cell
attach	Attach the file (from the local file system) that is specified in the **Description** cell.

bcc	"Blind copy" the message (the recipient specified in the **to** step does not see these addresses in the message) to the address specified in the **Description** cell. In the **Description** cell, specify an email alias or a semicolon-separated list of email addresses. Field replacements are supported.
cc	Copy the message to the address specified in the **Description** cell. In the **Description** cell, specify an email alias or a semicolon-separated list of email addresses of the recipients of the email message. Field replacements are supported.
from	Specifies the sender of the email message. Required for each mail session in a test case. In the **Description** cell, specify the email address of the sender. Field replacements are supported.
message	Adds content to the body of the email message. Type the message into the **Description** cell. To supply multiple lines of text, click **Advanced** for the **Command** property and type the text into the **Command** text box. Field replacements are supported.
open	Opens a new mail message for building and sending. Added automatically when you save a mail session as a procedure. (See the **send** action.) In the **Description** cell, select **application://com.fnfr.svt.applications.mail**.
reset	Clear the following fields in the email message that is currently being built: attach bcc cc from subject to
send	Added automatically when you save a mail session as a procedure. Sends the current email message. (See the **open** action.)
subject	Specify the text that should appear in the Subject line of the mail message. Type the text into the **Description** cell. Field replacements are supported.
to	Specifies the recipients of the email message. Required for each mail session in a test case. In the **Description** cell, specify an email alias or a semicolon-separated list of email addresses of the recipients of the email message. Field replacements are supported.

write	Appends the content that appears in the **Description** cell (the value of the **Command** property) to the body of the message and then moves to the next line in the message body. Because this action does not move to a new line, subsequent **write** or **writeline** actions are added directly to the end of the text. Type the message into the **Description** cell. You can use as many **write** and **writeline** actions as needed while building a message. Field replacements are supported. For example, use a **response** command in a field replacement to append the response for a step to the email message contents. **To create multi-line text** On the **General** properties page, click **Details**. Type the text into the **Command** text box.
writeline	Appends the content that appears in the **Description** cell (the value of the **Command** property) to the body of the message and then moves to the next line in the message body. Type the message into the **Description** cell. You can use as many **write** and **writeline** actions as needed while building a message. Field replacements are supported. For example, use a **response** command in a field replacement to append the response for a step to the email message contents. **To create multi-line text** On the **General** properties page, click **Details**. Type the text into the **Command** text box.
ContentType	Indicates the type of data to be included in the body of the mail. Mail session supports two types of data: **text/plain**, t**ext/HTML** Select **text/HTML**, the recipient of the message sees HTML message body. Select **text/plain**, the recipient of the message sees plain text message body.
insertImage	Adds image as specified in the description cell, to the message body. The images will be included as embedded/inline images within the message (text/HTML) content. Use the **insertImage** command to add image from your local system or your iTest project in test-case editor. • **From the message body dialog** (see "Send Email - Message body dialog" on page 909)**:** Right-click on the message body dialog, select the **Insert Image** menu option, and then select the required image. • **From the test-case editor:** select the **insertImage** command dialog of write/writeLine steps. .

3 Add other steps to configure other aspects of the message, like the **subject** and **message** contents. The steps can occur in any order and can be interspersed with other test case steps. You can use any number of **attach**, **write**, and **writeline** steps.

4 Add a **send** step at the appropriate location to send the current email message as configured up to that point.

Constructing and sending multiple email messages from a test case

To simultaneously build more than one message in a test case, use the **Session** property to identify steps associated with a particular message. For example, you can associate all steps for one message by setting Session to **debug_mail** and the steps for a different message by setting **Session** to **traffic_trend_mail.**

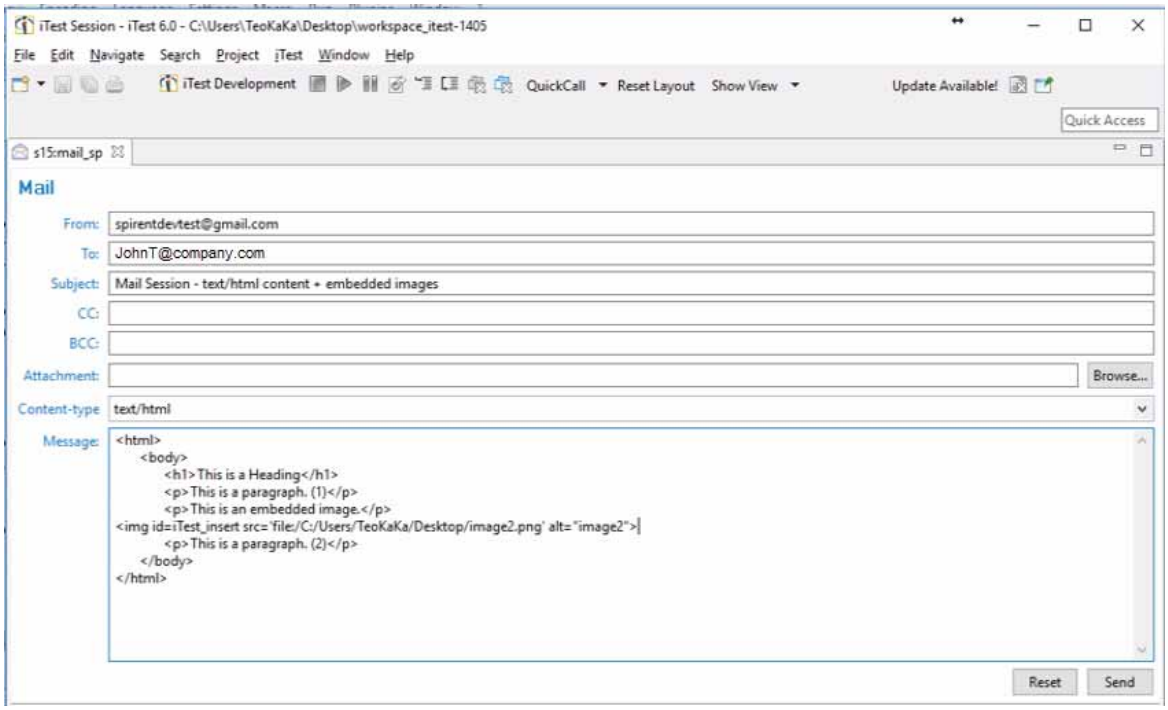
Storing response data in a variable

See "Storing a response into a variable (for use later in the test)" on page 140.

Sending an email message manually by defining a new session profile and then starting it

1 When you click **Start** on the **New Session** tab (**Start** page) of the session profile editor, an email message form opens. Fill in as many fields as needed. To actually send the message, you'll need to specify at least the **To** and **From** settings. For basic instructions on configuring a session, see "Session profiles: Session configuration settings" on page 71.

Send Email - Message body dialog



2 Click **Send** to send the message. iTest now captures all Mail session settings as steps.

Typically, you'll save the Mail session as a procedure, as shown earlier in the example. When you add the procedure to a test case, you can edit the properties of the message. For example, you can use the **Write** action to place test results into the message body. You might also alter the **to** setting to send failure results to one email alias and pass results to another alias.

To construct and send multiple email messages

To simultaneously build more than one message in a test case, use the **Session** property to identify steps associated with a particular message. For example, you can associate all steps for one message by setting Session to **debug** and the steps for a different message by setting **Session** to **traffic_trend**.

Sending Pass/Fail email messages

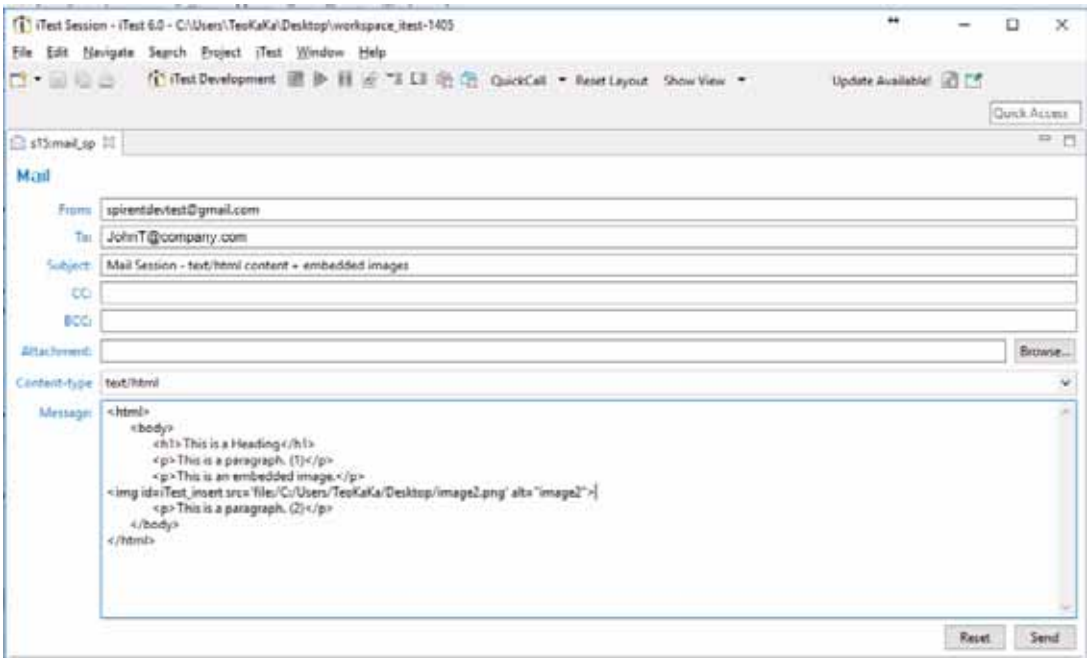
In addition to the email messaging that you configure here, you can use the test case Email properties to specify the conditions under which iTest will send a Pass/Fail email message upon completion of the test case.

Defining a Mail (SMTP) session

Mail sessions differ from typical sessions with devices. The reason for creating a Mail session is to enable your test cases to send email that can include test case response data and pass/fail results. When you start a Mail “session”, you are really preparing an email message that you can copy into a test case as a procedure. You can then add Mail **write**, **writeline**, and **message** steps that add response data to the message. A **send** step then sends the email. This topic provides instructions on configuring the session profile file that you use to start a session.

1 Here's an example Mail (SMTP) Session window. Prepare the email message as test cases should send it.

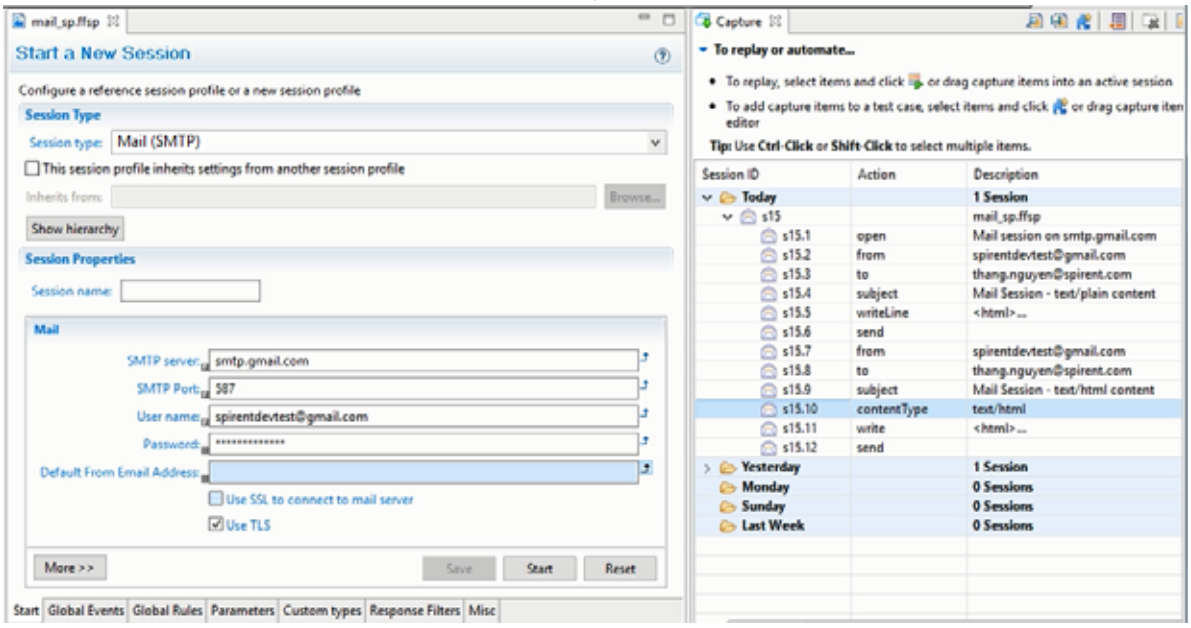
You can add plain text here, but once this message is added to a test case as a set of Mail session steps, you can use field replacements to insert test data into the Subject or Message, add a file as an attachment, or even change the mail sender (From) or recipient (To).



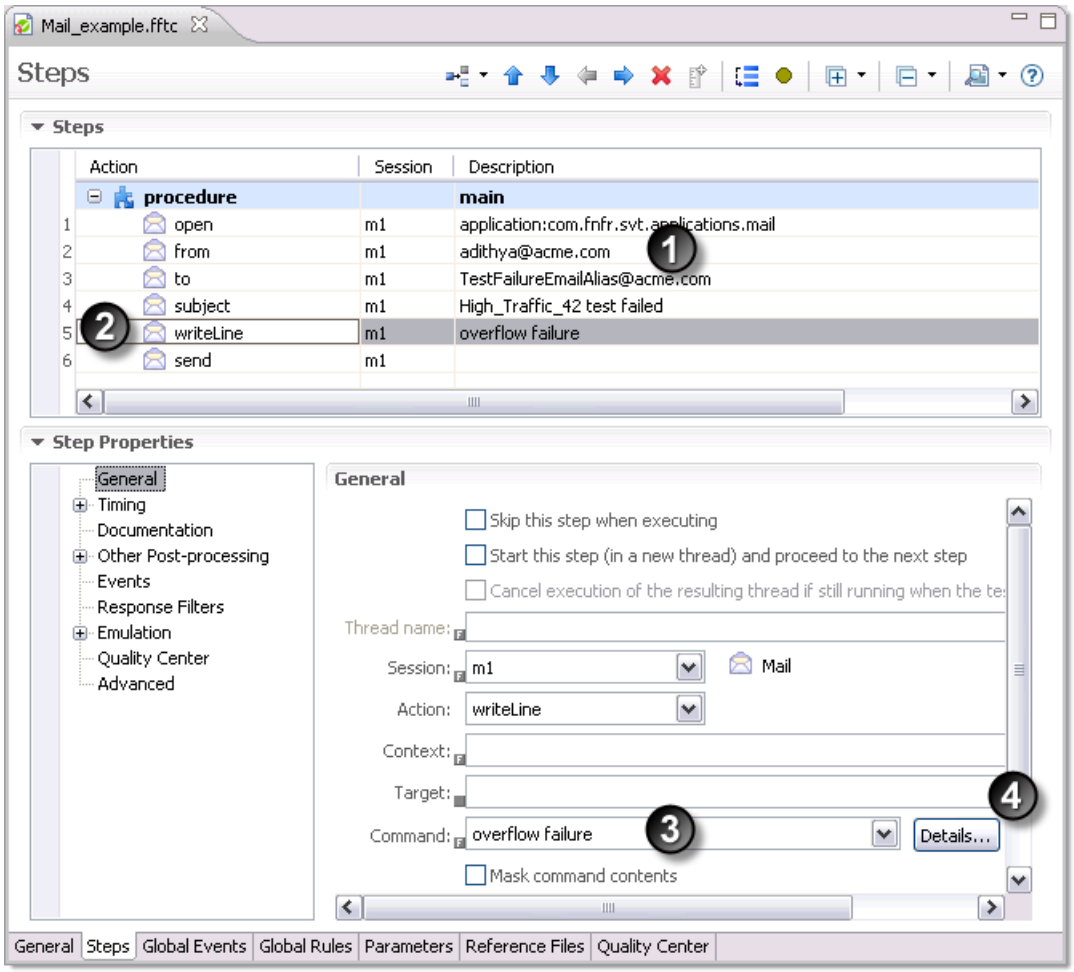
2 When you click **Send**, two things happen:

- The SMTP server sends the email
- iTest captures the contents of the message as a group of Mail session steps.

You can now save the steps into a test case.


mail_sp.ffsp
Start a New Session
Configure a reference session profile or a new session profile
Session Type
Session type: Mail (SMTP)
This session profile inherits settings from another session profile
Inherits from:
Browse...
Show hierarchy
Session Properties
Session name:
Mail
SMTP server: smtp.gmail.com
SMTP Port: 587
User name: spirentdevtest@gmail.com
Password: ••••••••••••••
Default From Email Address:
Use SSL to connect to mail server
Use TLS
More >>
Save
Start
Reset
Start
Global Events
Global Rules
Parameters
Custom types
Response Filters
Misc
Capture
To replay or automate...
To replay, select items and click or drag capture items into an active session
To add capture items to a test case, select items and click or drag capture iten
editor
Tip: Use Ctrl-Click or Shift-Click to select multiple items.
Session ID
Action
Description
Today
1 Session
s15
mail_sp.ffsp
s15.1 open Mail session on smtp.gmail.com
s15.2 from spirentdevtest@gmail.com
s15.3 to thang.nguyen@spirent.com
s15.4 subject Mail Session - text/plain content
s15.5 writeLine <html>...
s15.6 send
s15.7 from spirentdevtest@gmail.com
s15.8 to thang.nguyen@spirent.com
s15.9 subject Mail Session - text/html content
s15.10 contentType text/html
s15.11 write <html>...
s15.12 send
Yesterday
1 Session
Monday
0 Sessions
Sunday
0 Sessions
Last Week
0 Sessions

Here's is the Mail procedure that results:



1. Here are the Mail session steps. Each part of the message is a separate Mail action.

2. To add response data collected during execution to the message, you'll add **write**, **writeline**, and **message** actions.

   The **send** step sends the message.

3. As with any test case step, you can modify the properties as needed.

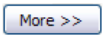
   For example, you would add field replacements here to include response data in the message body.

4. Click **Details** for multi-line contents.

Session profile property settings for Mail (SMTP) sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document with the appropriate settings, then you do not need to configure the settings again. See "About property settings" on page 72.

For Mail session profiles, there are no additional properties that appear when you click 

on the **New Session** page (the **Start** tab of the Session Profile editor).

Mail session profile properties

SMTP Server	Check with your system administrator for your SMTP server address. Enter either the DNS name or IP address.
SMTP port	Check with your system administrator for the port for the SMTP Server. The standard SMTP port and default setting is **25**.
User name / Password	Specify the username and password of the email account that will send the messages.
Default 'From.' email address	Specify the email address that should appear as the email sender for the message.
Use SSL to connect to mail server	Check with your system administrator to see if SSL is required when connecting with the server. Check the box to use SSL.

CHAPTER 47

Mail (POP3) Sessions

iTest supports POP3 protocol. You may use the Mail (POP3) sessions to retrieve emails from subscribers, view content, extract the required text and attachments, save the attached images as individual files, and then insert them into other sessions, for example, Selenium.

Receiving email messages during test execution

You can add steps that retrieves email messages with attached images (also embedded images) during execution. A test case can construct and retrieve as many email messages as required.

- Specify the number of message to retrieve from a mail server
- Specify the number of lines and to retrieve from the message. You may also retrieve specific lines from the message.
- Retrieve attachments (also in-line images, i.e., base64 encoded attachments) and save them as individual files to the specified location.

Preparing to retrieve email: Configuring the session profile

Before you use Mail steps in a test case, it's easiest to specify the POP3 server and user credentials in a Mail (POP3) session profile. You then refer to the session profile in the Mail (POP3) **open** step.

Retrieving email messages from a POP3 Server

The easiest way to steps to retrieve messages and/or images to a test case is to save a captured Mail (POP3) session as a procedure in the test case. You can then edit the Mail (POP3) session steps as described in the following sections (for example, to retrieve mails messages or attached files). See “Defining a Mail (POP3) session” on page 919.

Here's an example Mail procedure that results from saving a captured Mail session as a procedure:

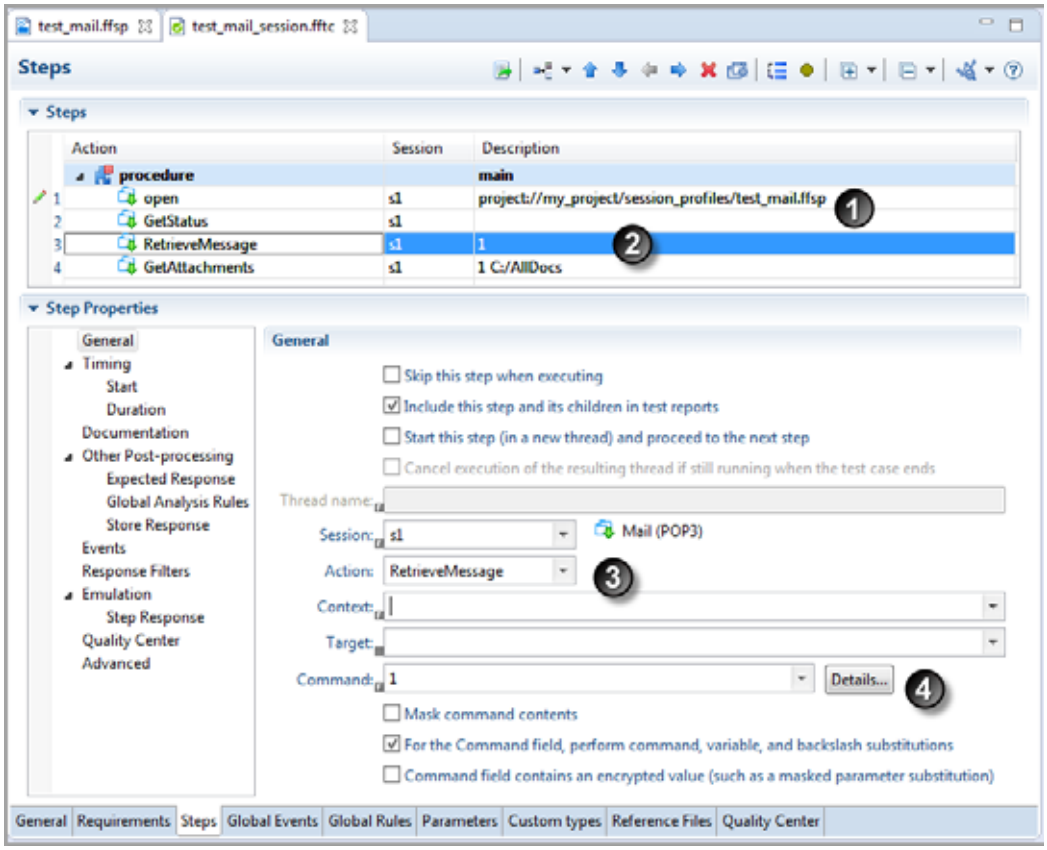
1. Here are the Mail session steps. Each commands is a separate command action.

2. To add response action performed during execution, you'll add **GetStatus**, **RetrieveMessage**, and **GetAttachments** actions.

3. As with any test case step, you can modify the properties as needed.

   For example, you would add field replacements here to include response data in the message body.

4. Click **Details** for multi-line contents.



Adding Mail steps manually

1. Add a step and select the **open** action. In the **Description** cell, select **application:com.spirent.itest.applications.popmail**. Session IDs for Mail steps do not interact with Session IDs of other sessions in the test case.

   Tip Add new steps by pressing **Ctrl+Space**, and then selection the command you would like to add.

2. In the **Action** cell, select a Mail action and, if needed, add information to the **Description** cell. Field replacements are supported for all commands. For example, Update description

will update the step command (Step Properties/General/Command), which supports Field replacements. See Chapter 28, “Field Replacements”.

Action commands for Mail (POP3) sessions

iTest Command	Command Description	Argument
Delete <messageID>	Delete a unique message ID The **Delete** command marks the message as deleted in iTest. However, the POP3 server does not actually delete the message until you **QuitSession**.	<messageId> - Message ID (mandatory).

Fetch	Get a summary of messages.
	Fetch [messageID]
	Example:
	`>LIST`
	`+OK 2 messages (320 octets)`
	`1 120`
	`2 200`
	`>LIST 2`
	`+OK 2 200`
	`>LIST 3`
		[messageId] - The message ID for which you would like to get a summary (optional).

GetAttachments	Get attachments and save them to the specified location. The command retrieves both the image file attached and the in-line images (base64 encoded images) and saves them as individual files to the specified location.	<messageId> - Message ID (required) <location> - Destination folder to save the attachments (required)
GetMessageUniqueID	Get a Unique ID for a message. GetMessageUniqueID [messageID]	[messageId] - Message ID (optional)

GetStatus	Displays the number of messages currently in the mailbox and the size in bytes.
	Example:
	`TotalSizes: 587890`
	`NumberOfMessages: 3`
	`Deleted: 0`
	`Result: OK`	

Login	Login <name> <pass or digest>
	Example:
	`S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>`
	`C: APOP mrose c4c9334bac560ecc979e58001b3e22fb`
	`S: +OK maildrop has 1 message (369 octets)`
		<name> - User name (required)
		<pass> - Password (required)
		<digest> - a MD5 digest string (required)
		- Use <pass> or <digest>
		If <pass> provided, use USER and PASS commands to login.
		If <digest> provided, use APOP to login.

iTest Command	Command Description	Argument

QuitSession	End the POP3 conversation
	Example:
	`C: QUIT`
	`S: +OK dewey POP3 server signing off (maildrop empty)`
	`...`
	`C: QUIT`
	`S: +OK dewey POP3 server signing off (2 messages left)`
	Note Any messages marked as deleted in iTest by the Delete command are deleted by the POP3 server only when to The QuitSession.	

ResetSession	Reset the session to its initial state. That is, if you have deleted messages via the **Delete** command, use **ResetSession** command to un do the delete action. This applies only before you **QuitSession**.
	Example:
	`C: RSET`
	`S: +OK maildrop has 2 messages (320 octets)`	

RetrieveMessage	Retrieve a particular message. and/or specific lines from the message.
	RetrieveMessage <messageID> [numberOfLines]
	Example:
	`>RETR 1`
	`+OK 120 octets`
	`<the POP3 server sends the entire message here>`
		<messageId> - Message ID (required)

	Retrieving part of the message, specify the number of lines of the message body you would like returned
	`Examples:`
	`C: TOP 1 10`
	`S: +OK`
	`S: <the POP3 server sends the headers of the`
	`message, a blank line, and the first 10 lines`
	`of the body of the message>`
	`S:..`
	`...`
	`C: TOP 100 3`
	`S: -ERR no such message`
		[numberOfLines] - a number of lines (optional)
		If <numberOfLines> has been provided, we will use TOP command. If not, use RETR instead.

1. Add any required command. The steps can occur in any order and can be interspersed with other test case steps. You can use any number of any of the steps listed above.
2. Add a **ResetSession** step at the appropriate location to reset to the session to original sate. Add a **QuitSession** step to exit the session.

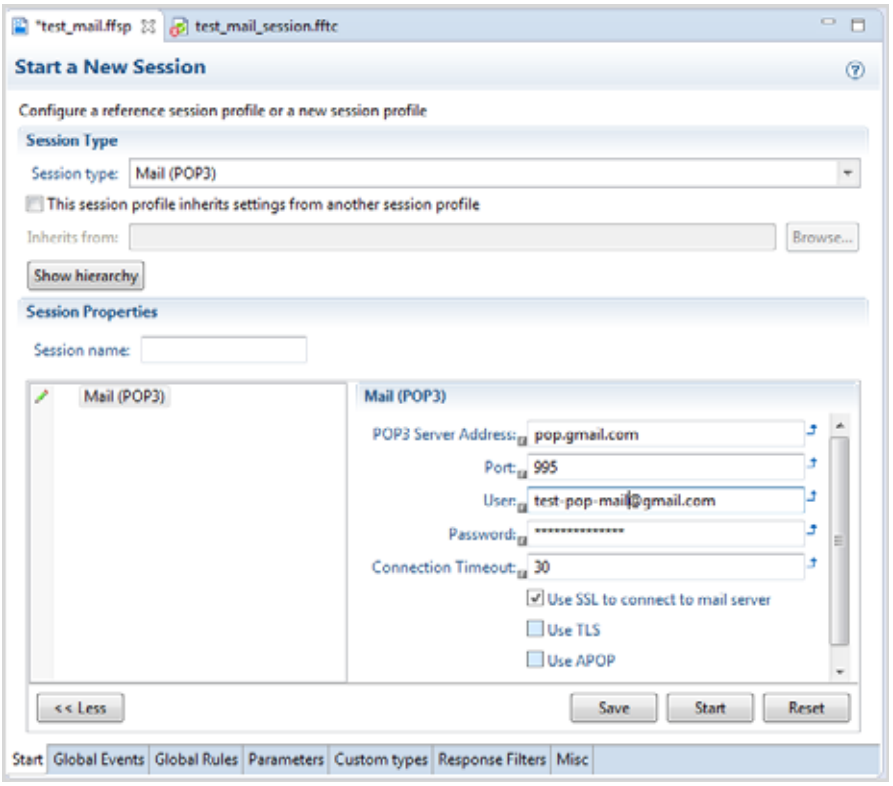
Storing response data in a variable

See "Storing a response into a variable (for use later in the test)" on page 140.

Defining a Mail (POP3) session

Mail (POP3) sessions differ from typical sessions with devices. The reason for creating a Mail (POP3) session is to enable your test cases to receive email that can include test case response data and pass/fail results. When you start a Mail (POP3) "session", you are specifying a POP3 email server and email address from where you can retrieve emails, copy email attachments to your work space. You can then add Mail **GetStatus**, **RetrieveMessages**, and **GetAttachments** steps. **GetStatus** retrieves the mail-box status and the number of emails in the mail-box. A **RetrieveMessages** step retrieves an email from the POP3 server. This topic provides instructions on configuring the session profile file that you use to start a session.

Here's an example Mail (POP3) Session window. Click Start and the **PopMail Console** opens.

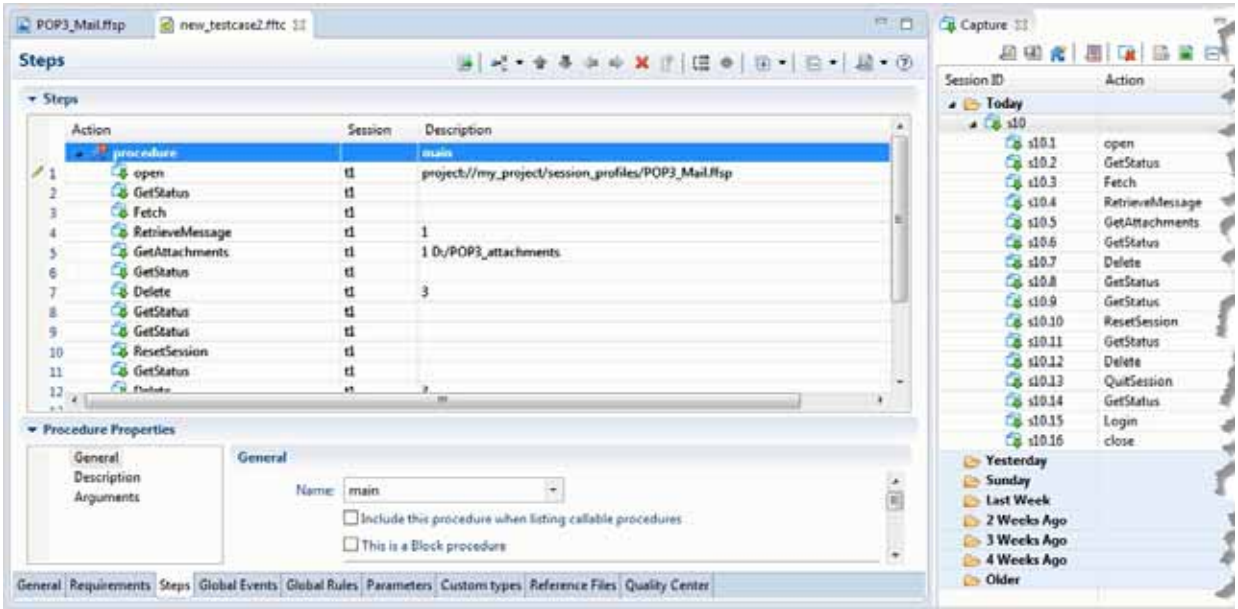


Note

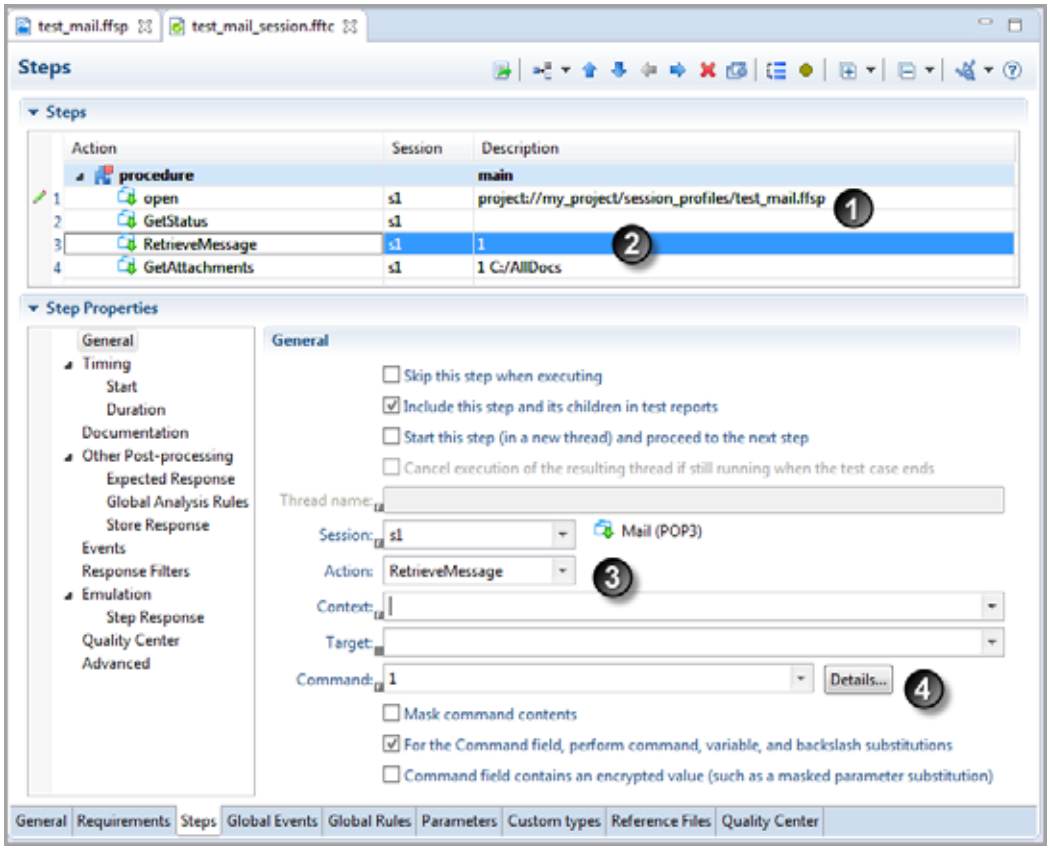
- Mail (POP3) port 110 is commonly used to connect without SSL and TLS options (default settings)
- Mail (POP3) port 995 is commonly used to connect with SSL option
- Mail (POP3) port 110 is commonly used to connect with TLS option

Perform the required task using the commands as required, as shown below. See also Mail (POP3) Session Commands for details.

iTest captures the tasks you performed (commands you used) as a group of Mail (POP3) session steps. You can now save the steps into a test case.



Here's is the Mail procedure that results:



1 Here are the Mail (POP3) session steps. Each part of the message is a separate Mail action command.

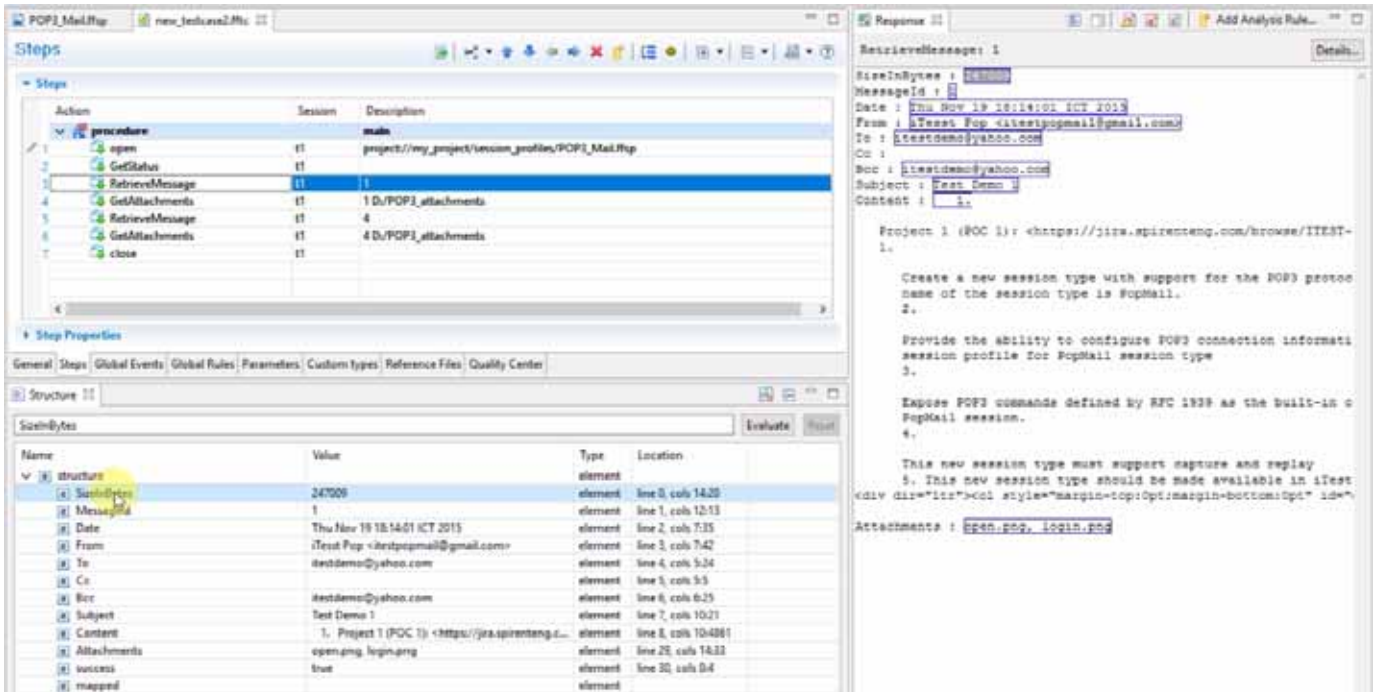
2 To add response data collected during execution to the message, add **GetStatus**, **Fetch**, **RetrieveMessage**, **GetAttachments** message command actions.

3 As with any test case step, you can modify the properties as needed.

For example, you would add field replacements here to include response data in the message body.

4 Click **Details** and add multi-line contents.

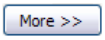
Based on the step response, you can add the analysis rule for any step in iTest. For example, after capturing mail session commands and running the test case, you may add analysis rule for any command, for example, RetrieveMessage.



Session profile property settings for Mail sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document with the appropriate settings, then you do not need to configure the settings again. See "About property settings" on page 72.

For Mail (POP3) session profiles, there are no additional properties that appear when you click 

on the **New Session** page (the **Start** tab of the Session Profile editor).

Mail session profile properties

POP3 Server	Check with your system administrator for your POP3 server address. Enter either the DNS name or IP address.
POP3 port	Check with your system administrator for the port for the POP3 Server. The standard POP3 port and default setting is **110**.
User name / Password	Specify the username and password of the email account that will retrieve the messages.
Connection Timeout	Specify the maximum time to wait to establish a connection to the PoP3 server.
Use SSL to connect to mail server	Check with your system administrator to see if SSL is required when connecting with the server. Check the box to use SSL.

CHAPTER 48

Process Sessions

Process session window: Running processes on the local computer

A test case step in a Process session type can start, run, and fetch the result and output of a process on the local computer (where the test case is running). Typical uses:

- Invoke and send commands to a command-line utility.
- Access Ethereal and extract high level packet information directly from the test case.
- Determine the operating system of the local computer (Windows or Linux) to decide whether to send backslash or forward slash characters in commands.
- Start a GUI application under test and then use other CLI session type steps to control its harness.

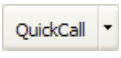
Process commands

iTest command	Description
cd	With no argument, **cd** returns the current directory. With a directory as the argument, changes the directory that will be used as the working directory when processes are started or run. This does not affect the working directory of current processes. cd cd directoryPath
kill ?*processID*?	• **kill** kills all processes that were started using the **start** command • **kill** *processID* kills the specified process (with the same device and session) that was started using the **start** command. A **kill** step is complete immediately. The response to a **kill** command contains a message indicating whether the process was killed or was already complete.

run	Starts a new process and waits for it to complete. iTest does not move to the next test case step until the process is complete. You have the option to use a Timeout Completion rule to limit run time. The application window appears on the screen unless you set the Create No Window property to True. In most cases (for example, GUI applications), set Create No Window to False (the default setting). Optional: Use the -q[uiet] argument before the command to display only command output in the response, and not the exit code text, typically: Process 1 terminate, exit code: 0 Response run accepts all STDOUT produced by a process. Windows applications might not display the same STDOUT in the window. All process STDOUT is returned in the response body. Any STDERR is appended to the response body with a leading `STDERR:` string. There are two response headers: Result and ExitTime. Result returns the integer code that the associated process specified when it terminated. ExitTime returns the time that the associated process exited. Arguments run applicationName applicationName is the file to be executed (which can be an executable or a file that the operating system can start because of a file association) and may optionally contain additional arguments. If the filename includes spaces, then it must be surrounded by double-quotes. (The arguments should not be surrounded by quotes.) The result and the output of the process are placed into the response for the run step.
show platform	Returns information about the operating system in the body of the response. Here's an example: Process>show platform name: Windows XP ver: 5.1 arch: x86 type: win32 The intended use of the show platform action is to determine the operating system of the local computer (Windows or Linux). Based on the result, you can decide whether to send backslash \ or forward slash / characters in subsequent commands.
show process	**show process processID** Returns process status information (running or terminated, exit code if terminated). **show process** Returns process status information for all processes started in the current session (running or terminated, exit code if terminated).

start	Starts a new process on the local computer. The test case step is complete immediately after starting the process. Typically, test case execution immediately moves to the next step. A process started with **start** that has not ended by the time test case execution completes is automatically killed. Arguments start applicationName The command cell for a start action contains the file to be executed (which can be an executable or a file that the OS can start because of a file association) and may optionally contain additional arguments. If the filename includes spaces, then it must be surrounded by double-quotes. (The arguments should not be surrounded by quotes.) Concurrent Sessions The device name and session fields for the start step (if any) associate this process instance with other Process actions that execute later in the test case. This enables you to start several processes that will run concurrently. Portability To make test cases portable, define a parameter that holds the filename of the file to be executed. The command can use a field replacement to populate the filename.
wait	Blocks the session window until the specified process terminates (by external action for example). wait processID

Tips for Process sessions

- Determining the operating system at runtime: If your test case does not know ahead of time which operating system it will use, then, at runtime, the test case can use a Process **show platform** command to return the value of the operating system running on the local computer. Subsequent steps can then use this information to specify either backslash \ or forward slash / characters in commands.
- You can set preferences for Process sessions. See “Preferences: Spirent > Sessions” on page 865 and “Setting iTest preferences” on page 861.
- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click 

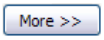
in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Session profile property settings for Process sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Process

Working Directory	Optional. The initial working directory to use for submitting commands when the process starts.
Use custom environment	When you uncheck the box, then the environment variables specified for the **Environment** property are passed to new processes. When you check the box, the current environment variables are passed to processes that are launched with **run** or **start**.
Environment	Optional. Applies only If you check **Use custom environment**, The environment variables that you specify here overwrite or supplement the current environment variables for launched processes. Type the environment variable settings as a list of name-value pairs, separated by semicolon or colon (; or :). All spaces are ignored. For example, `var_name=value; var2=value2` **The PATH variable for batch executables** You cannot use the **Environment** property to specify a PATH variable (for example, **PATH=D:/my_batch_path/**) to enable you to run executables in that path. To run such an executable, you must specify its full path. If your System PATH variable contains the path to a folder (for example, D:/path/) that contains batch executables (for example, test.bat), you can use commands like **run test.bat** or **start test.bat** in Process sessions. You cannot use commands like **run test** or **start test**.
Kill running processes after session end	Optional. Uncheck the box to allow running processes to persist after the Process session ends. This setting typically applies when you use the Process **start** action.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Example Process session on Microsoft Windows

```
Spirent Process command interpreter.
Copyright (c) 2005 - 2011, Spirent Communications, Inc.

process>help
     cd                     - Show current working directory
     cd <working directory>         - Change working directory
     exit                    - Exit process application
     help                    - Display application commands
     help <prefix>              - Display application commands
     kill                    - Kill all running processes
     kill <process ID list>         - Kill the process
     run <command>              - Start the process and wait for termination. Use
-q[uiet] to display only command output
     show platform              - Show platform information
     show process               - Show information about all processes

     show process <process ID list>    - Show process information
     start <command>             - Start the process
     wait <process ID>            - Wait for a process termination

process>show platform
   name: Windows XP
   ver: 5.1
   arch: x86
   type: win32

process>start notepad
Process started, ID: 0

process>show 0
Invalid arguments. Type "show ?" for a list of subcommands.

process>show process 0
ID | Command                              | State   | Exit Code
-------------------------------------------------------------------------------
0  | notepad                              | RUNNING | N/A

process>start cmd
Process started, ID: 1

process>show process 1
ID | Command                              | State   | Exit Code
-------------------------------------------------------------------------------
1  | cmd                                 | RUNNING | N/A

process>kill 1
Process 1 terminated, exit code: 1

process>wait 0
Waiting for process 0 to terminate...
```

```
Process 0 terminated, exit code: 0

process>cd
Working directory: 'c:\'

process>cd c:\temp
process>cd
Working directory: 'c:\temp'

process>exit
```

Setting preferences for Process sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Process**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Process

Number of allowed simultaneously running processes	Specify how many processes can run on the local computer at the same time.

CHAPTER 62

REST sessions

Working in the REST session window

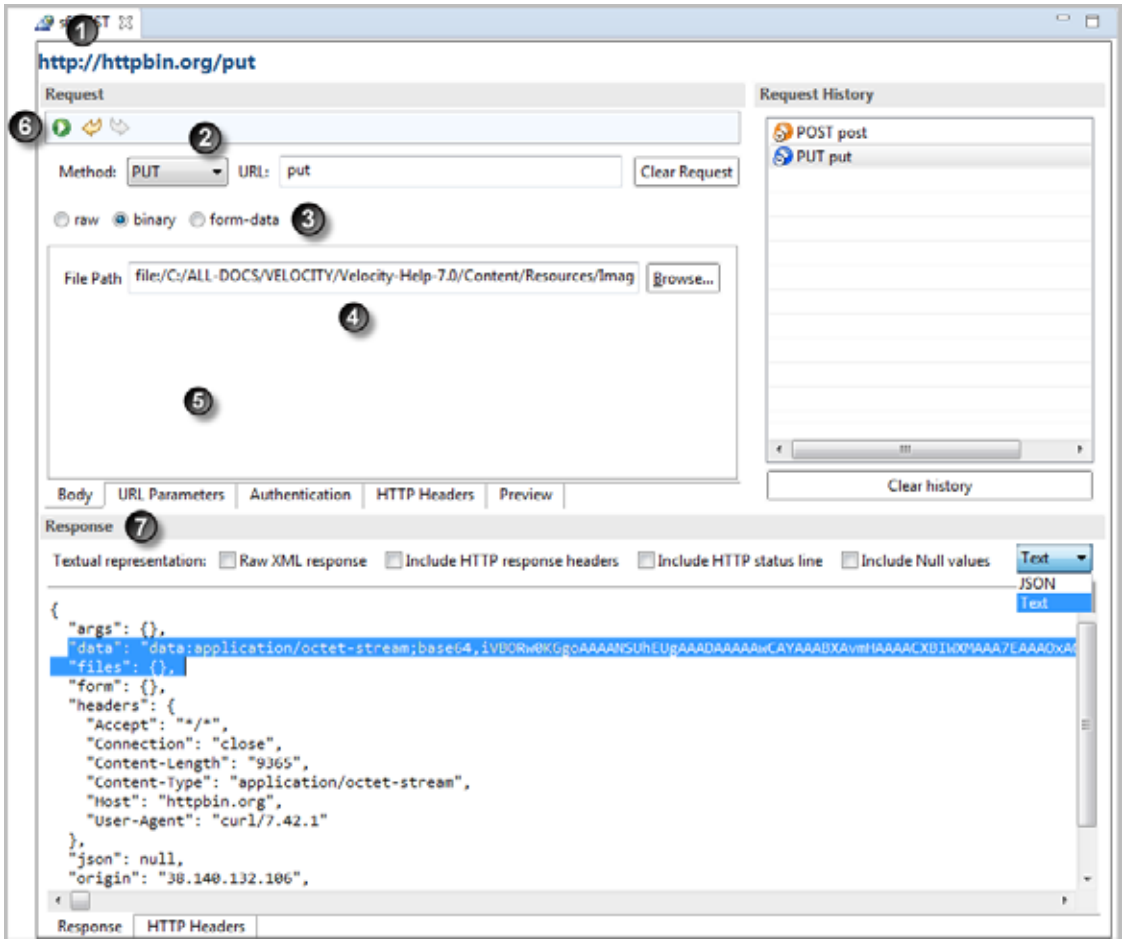
The REST session window, in addition to providing a work surface for composing and submitting HTTP requests, add From Data entries (text or file to be sent to server), and upload files.

Any request that you submit in the iTest session is forwarded to the specified RESTful service. The service performs the action and returns its normal response. You can view the response in the **Response** section of the session window.

iTest captures all of the actions that you perform in a session and all of the responses. You can use the captured items to create test case steps that interact with the REST server.

Example REST session

In this example REST session in iTest, we submit a PUT request, and on the **Body > Binary** tab, select a file to upload to the server. For such a request, the server responds by echoing the values that we sent in the request.



1 **Request URL**

The URL is updated appropriately whenever you modify a parameter or an HTTP header.

You specify a default server URL in the session profile. You can modify the URL as needed here. You have the option to specify a relative URL (like "**../path/**") or an absolute URL (like **http://hostname/**).

2 **Specify the HTTP request method**

iTest supports **GET**, **POST**, **PUT**, **DELETE**, **HEAD**, **OPTIONS**, **TRACE, and PATCH**.

3 **Select Data Type option: Raw, Binary, or Form-Data on the Body tab**

- **Raw** tab: Allows you to enter the text that will be echoed in the response.
- **Binary** tab: Allows you to browse to the location of the file you wish to upload.

Note The **Binary** and **Form-Data** options are available only for the **POST** and **PUT** methods.

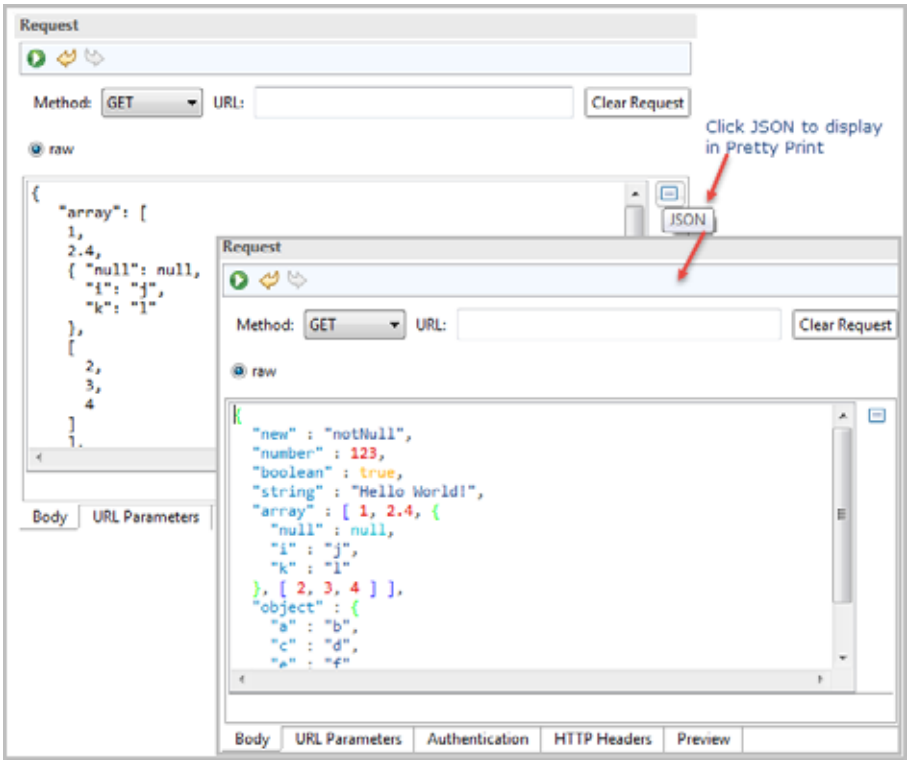
- **Form-Data** tab: Select types of the entries: **Text** or **File**. Add entries, select the **Form-Data** File type to upload files on to the servers.

4 Select Body tab > Raw (Read/Write view)

Enter the required text. Click **JSON** button on the right-hand side of the window. The text content displays in pretty print, if the text is a valid **JSON** syntax.

If the text entered is not a valid **JSON** syntax, no changes are made to the text layout and a message indicates that the text does not include valid **JSON** syntax.

You may edit text and click the **JSON** button to display the content in pretty print.



5 Select Body tab > Raw (Read only view—Content displayed in the REST session Response area)

See Step 8 and Step 9.

6 Select Options Binary or Form-Data on the Body tab

Select Binary tab option (on the Body tab) and Upload file

Click the browse button to navigate to the file location, and select the file to upload. The file name and path selected will be populated in the File Path field.

- **Browse:** Click Browse, select a file to upload. You may select file from the local drive or from your iTest project.

Note You may select only one file at a time to upload.

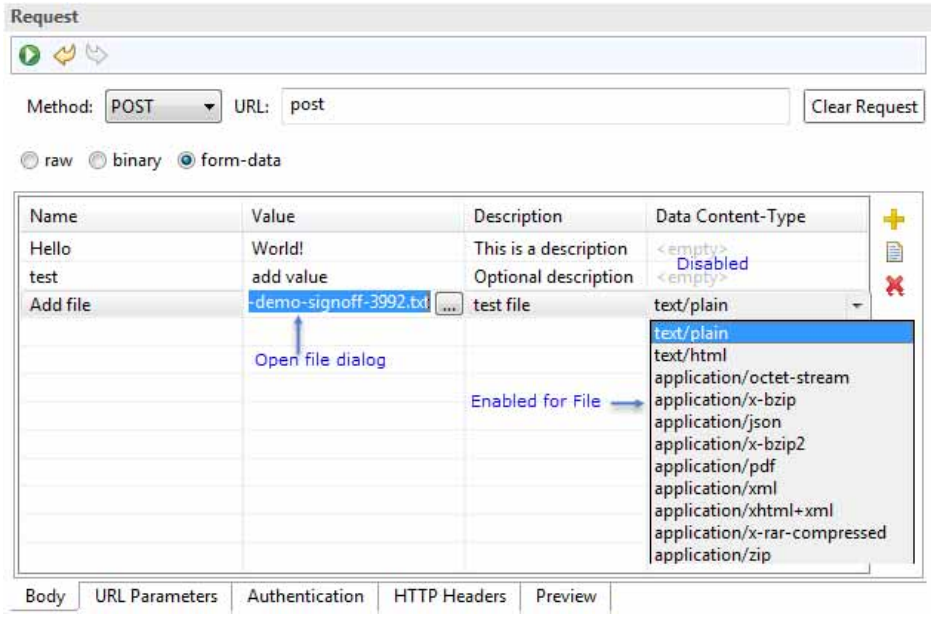
Go to **HTTP Headers tab,** specify the **Content-Type** header, for example, **application/json**. If you do not specify any **Content-Type** in the **HTTP Headers** tab, **Content-Type:application/octet-stream** will be added and used by default.

- **File Path:** The file path will be populated with the location of the selected file (local file URI or project URI), depending on your selection.

Select Form-Data tab option (on the Body tab)

Click **Add Text** or **Add File** and enter relevant information as described in the table below.

Name	Mandatory. Specify a name for the filed or the file name. When adding a file, the name is the name/id of an upload file control in an HTML form. By default, this field contains the file location.
Value	Mandatory. Specify the value for the text field or enter the name of the file to be sent. **Note** Browse to the location of the file and select file to be sent.
Description	Optional: Enter a description of the text field or the file being sent. This description is not sent to the server but is available from iTest.
Data Content Type	Optional. Specify the content type for the file to be uploaded. By default, the field contains an empty value. Select the content type from the dropdown list. **Note** The Content Type is available when **Add File** is selected and disabled when **Add Text** is selected.(displays <empty text>).



Important The content type is MIME type, associated by type of documents.

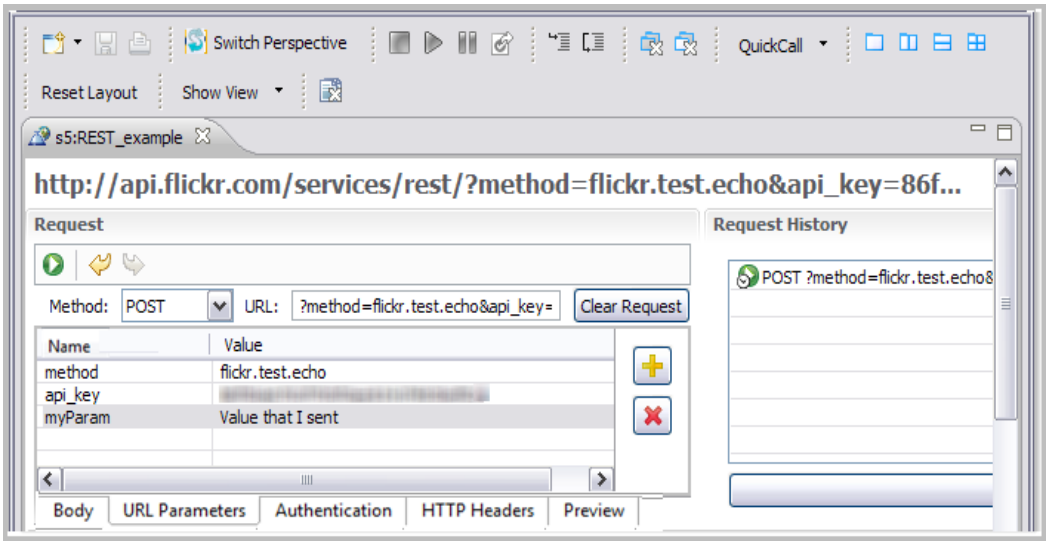
If a **Content-Type** is already specified as an **HTTP Header**, then the already specified HTTP header will be used. If not, the **Content-Type: multipart/form-data** will be used by default.

7 About the Request tabs

- Use the **Body** tab to paste or type an HTTP request body.

- Use the **URL Parameters** tab to specify parameters of the request.

  On the URL Parameters tab: specify Parameter Name and Value



  Use the **URL** tab to specify name appropriate for the RESTful service by typing or pasting parameter names and values. You can modify parameter name by typing directly in the **URL** text box. Use the tools to add or delete parameters and undo/redo. The example, specifies values for two default parameters.

  You have the option to specify parameters and their values in the session profile.

- Use the **Authentication** tab to specify new authentication settings (described in "Authentication" on page 1192).

- The **HTTP headers** that you specify in the session profile are included with each request URL. Use the HTTP Headers tab to add headers to the current request URL.

  If you selected Binary in the **Body** tab, selected a file to upload, and did not specify the Content-Type header in HTTP Headers tab, **Content-Type:application/octet-stream** will be added and used by default.

- Use the **Preview** tab to see the whole HTTP message

  For example:

  PUT put HTTP/1.1

  Content-Type: application/json

  If you specify the name and value on the URL tab, the request portion of the URL is updated appropriately whenever you modify a parameter or an HTTP header.

8 Send the request

Click **Submit** . Requests that you submit are saved in the **Request History** section, so you have quick access for easy reuse (select the request, make any required changes, and then click).

If we were to save this captured interactive session as an iTest test case, this request would be saved as a step with an **Action** of **PUT** and the **Description** cell for the step would hold the request Method, put.

For example (see On the URL Parameters tab: specify Parameter Name and Value), if the URL tab is used to specify a RESTful service and the request method is POST, then the request would be saved as a step with an **Action** of **POST** and the **Description** cell for the step would hold the request URL.

Note In test cases, all REST steps support field substitution, so you can dynamically modify the request URL at test case runtime. More detail at "REST test cases" on page 1189.

When you submit the request, you may be prompted for authentication information.

- Provide login/password if HTTP basic authentication is enabled on the server
- Select a client certificate for HTTPS

9 Response

The server response echoes the data/file/values sent in the request.

Note iTest auto-maps responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps and can immediately write analysis rules that use the auto-generated queries.

The **Response** section of the REST session window displays the response from the server in plain text and in several other representations. Regardless of the way you choose to view the response on this page, iTest always captures the full XML response and uses a structured XML version for analysis in iTest.

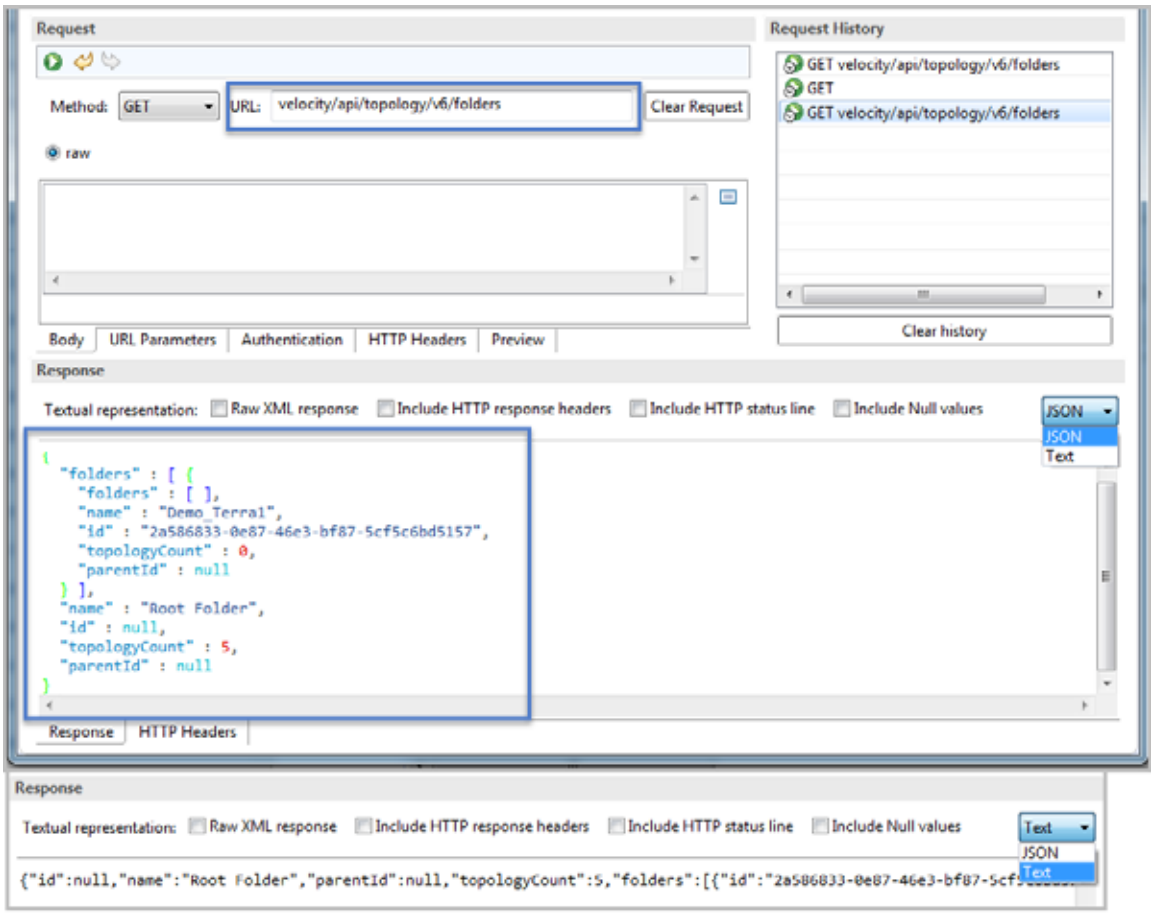
On the **Response** tab, the default display format is auto-detected as **JSON** or **Text** form.

- A text representation of the values are returned in the response (a single value, a set of name/value pairs, or a table).

- If **JSON** syntax is detected, iTest displays text formatted as **JSON** pretty-print.

  If **JSON** format is not detected, the data will be displayed as TEXT and will interpret/present the data accordingly.

  Click **JSON**/**Text** options from the dropdown list on the session **Response** Window to toggle the response view display as **JSON** Pretty-Print or **Text**.

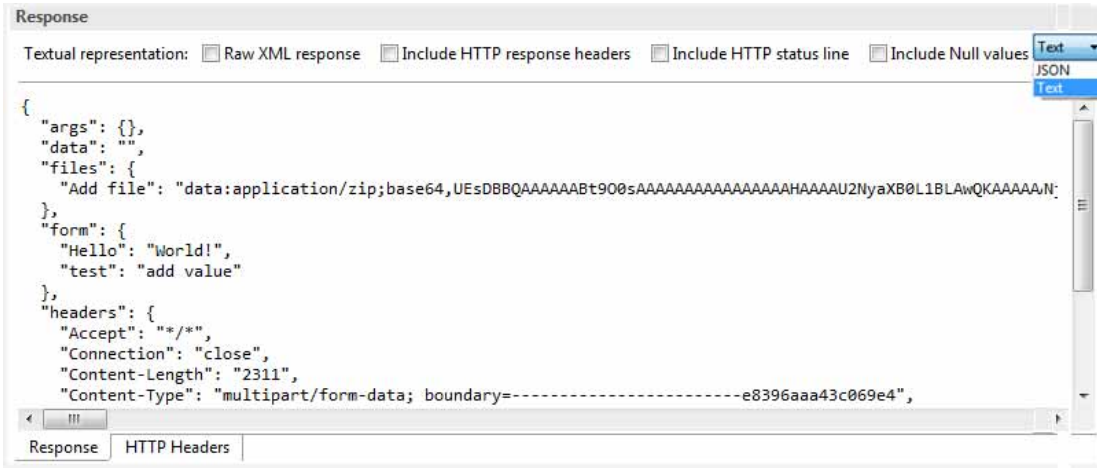


When iTest auto-detects **JSON** format, or you click the **JSON** button, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see "Setting preferences for JSON Pretty Print" on page 252 in Chapter 9, "JSON Editor").

Other Formatting options:

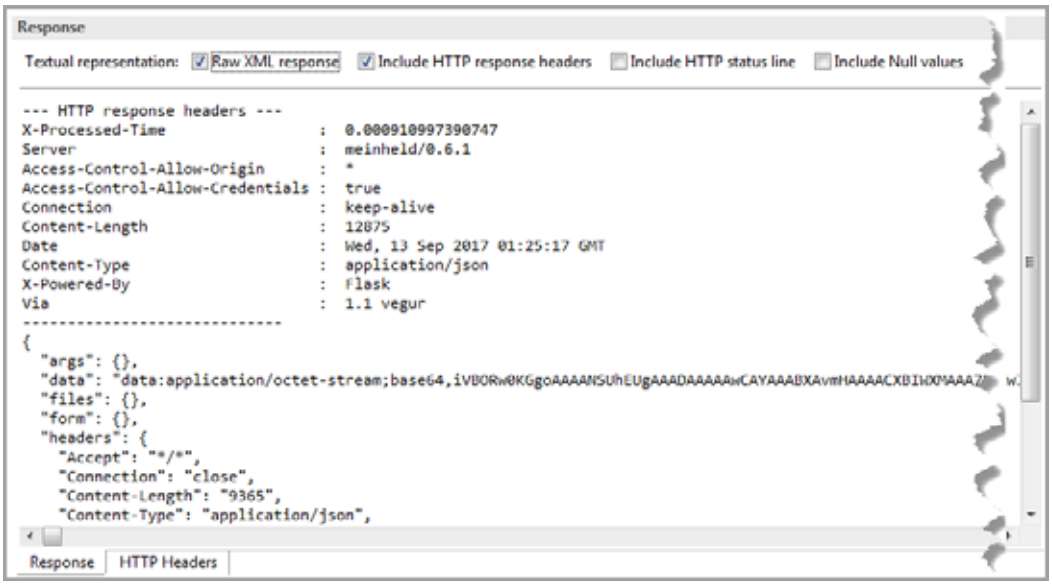
Below are the other formatting options (you must resubmit a request to view the response in a different display format):

- On the **Response** tab, check **Raw XML response** to view the full XML text of the response

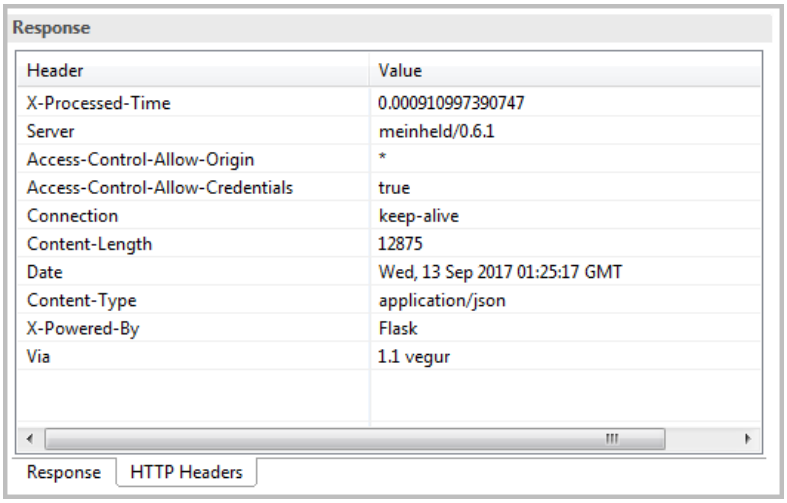


- On the **Response** tab, check **Include HTTP response headers** to display the header text from the response.

  You may also click  to display the available header templates and select the required template from the list, and modify as needed after inserting. See "HTTP Header Templates" on page 1193 for creating and maintaining header templates.



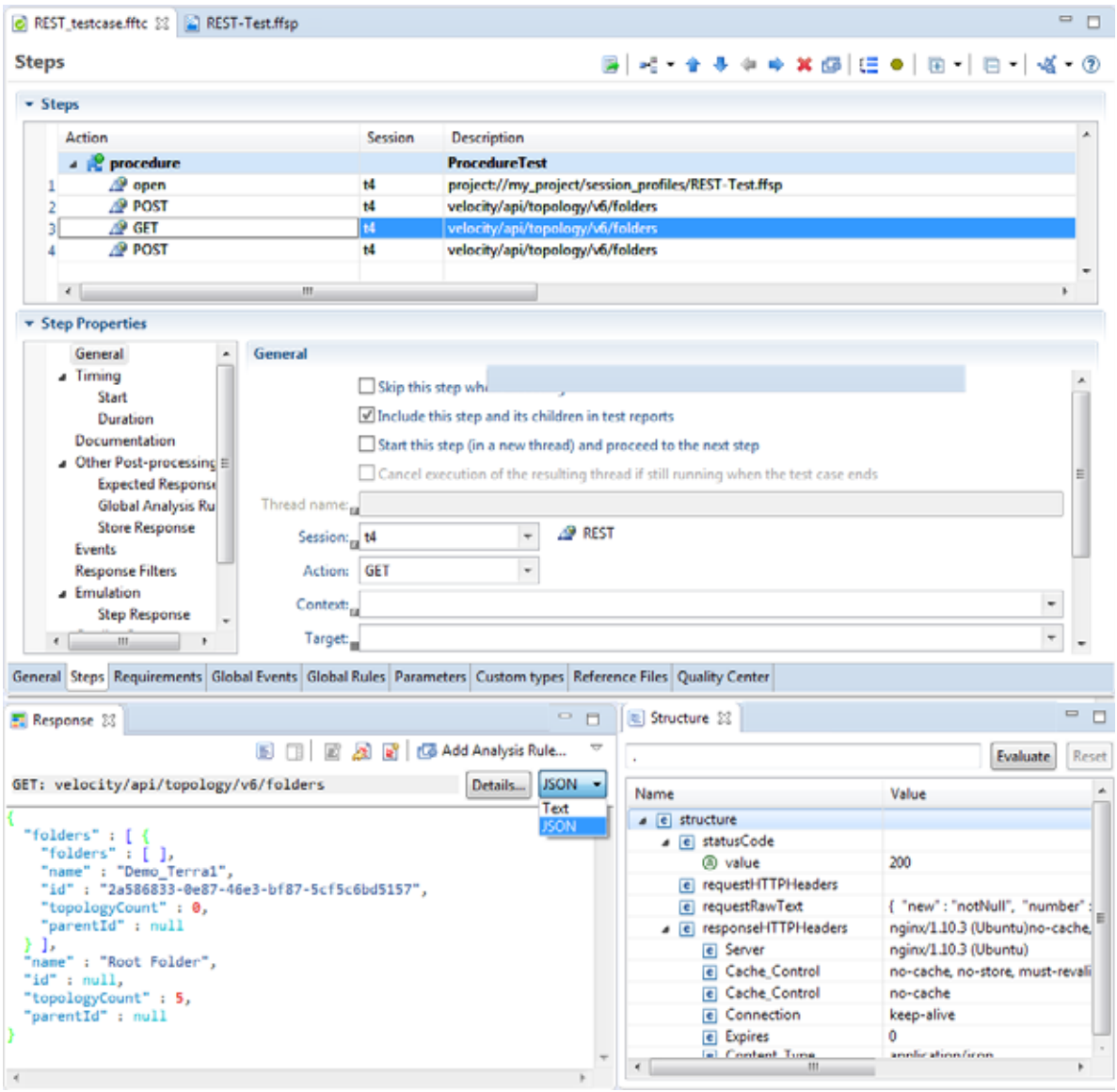
- On the **HTTP Headers** tab, the response header appears in a table



Note In another example, we might have used a iTest REST session to enable a port on a router. After enabling the port, we might open a iTest SSH session with the router to execute a verification step that checks the status of the port that the REST server says had been enabled.

REST test cases

When you save a captured interactive session as an iTest test case, each request method is saved as an **Action** step. In addition, The response is auto-mapped (iTest auto-generates a named query for each data value in a response).



In the Test Case Response View, the default display format is auto-detected as **JSON** or **Text.**

- **JSON** format detected is rendered as Pretty-Print.

  If **JSON** format is not detected, the data will be displayed as **TEXT** and will interpret/present the data accordingly.

Click **JSON**/**Text** options from the dropdown list on the test case **Response** window to toggle the response view display as **JSON** Pretty-Print or **Text**. When iTest auto-detects **JSON** format, or select the **JSON** option to display the content, the pretty print format is assigned as per your settings (see “Setting preferences for JSON Pretty Print” on page 252 in Chapter 9, “JSON Editor”).

Modifying the request URL at runtime

In test cases, all REST steps support field substitution, so you can dynamically modify the request URL at test case runtime.

Modify or Add Action commands

REST test cases support modifying the selected Action command. For example, as shown in the screenshot above, follow these steps.

- Select **POST** in the **Action** section.
- **S**elect **REST POST Properties > POST Step Properties** (in the **Step Properties** section).

On the **POST Step Properties** window, the **Body Data Type**, **Message**, **File Path** displays the properties captured.

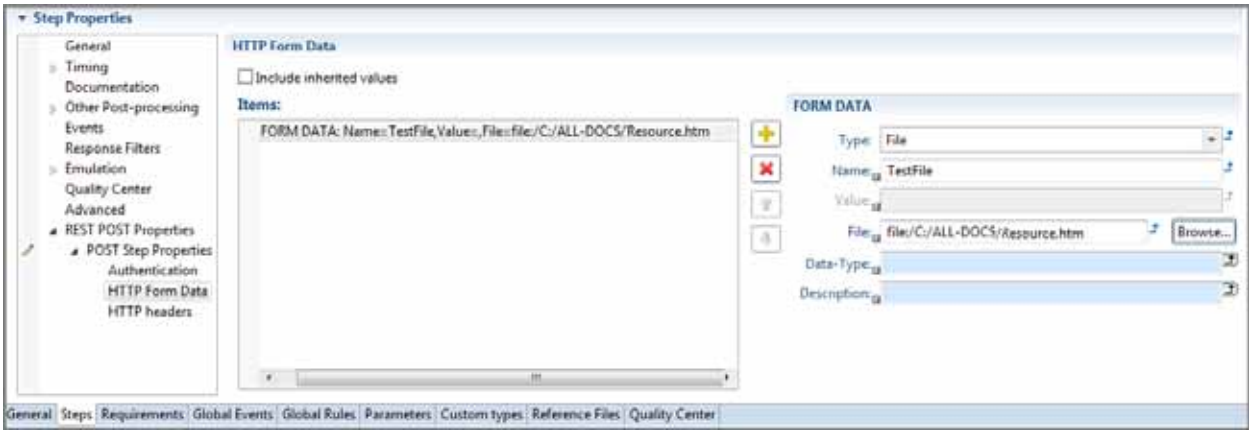
The Body Data Type is identified as **Raw**, **Binary**, or **Form** as specified during capture (use the drop-down to modify the data type).

- If **Data Type** is **Raw**: The **Message** field will be enabled and the **File Path** field will be disabled.
- If **Data Type** is **Binary**: The **File Path** field will be enabled and the **Message** field will be disabled.
- If **Data Type** is **Form**: Both the **Message** filed and **File Path** field will be disabled.

Note The Form and Binary options are available only for POST and PUT methods.

Select **REST POST Properties > POST Step Properties> HTTP Form Data**.

The **HTTP Form Data** window lists the Form Data page as shown below.



Each item has these properties: Name, Value, Description, Data Content Type and Type.

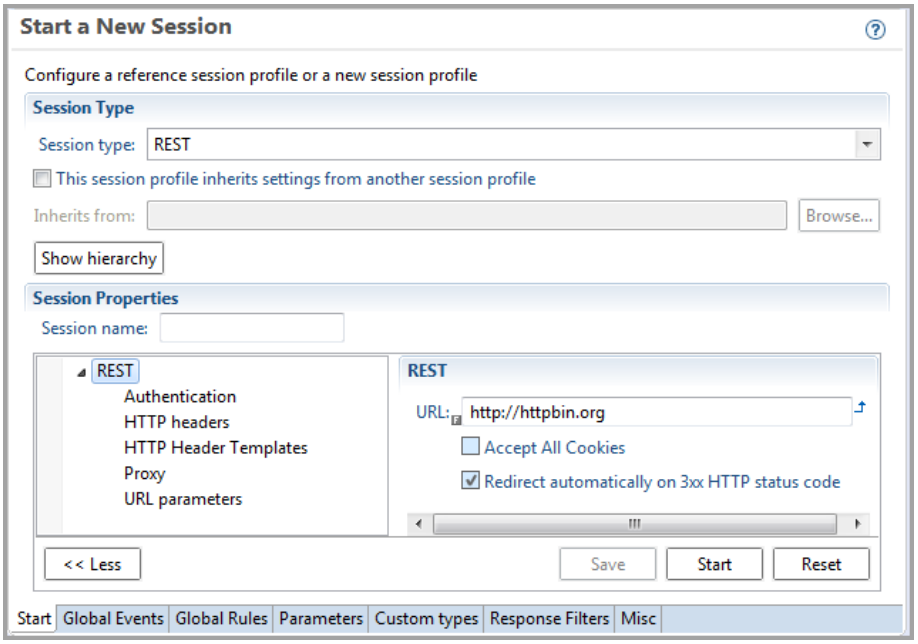
You may select **Include Inherited value** or Add information in the **Form Data** as described in the REST Session Editor **Form-Data** tab on page 1183).

Session profile property settings for REST sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.



REST

URL	URL of the RESTful service. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html** **Note** You can specify a different URL on the session window while preparing a request.

Authentication

Authentication	Specify authentication: **None**, **Basic**, or **Secure** Default: **None**
User	Required if **Authentication** is set to **Basic** or **Secure**. Specify the user ID for basic HTTP authentication.
Password	Required if **Authentication** is set to **Basic** or **Secure**. Specify the password for basic HTTP authentication.
Keystore file	Required if **Authentication** is set to **Secure**. Specify the path to the client key file. Specify a URI that starts with **file://** or **project://** **Note** On Windows platform if no **Keystore file** is provided you may see the following error in the Error Log when sending HTTPS requests: `curl: (60) SSL certificate problem: unable to get local issuer certificate` This indicates that iTest cannot find CA for the server certificate. To ignore the error, go to the **REST** session profile > **Authentication** and select "**Trust any SSL certificate from the server**". If you see this error using a key storage for Secure Authentication, that probably means that your storage doesn't contain the whole certificate chain.
Keystore password	Optional if **Authentication** is set to **Secure**. Specify the password to use to access the private key in the key file. The text is masked here and in all locations in which it is used.
Trust any SSL certificate from the server	Check the box to cause iTest (running as the REST client) to trust any SSL certificate coming from the server. Default: unchecked

Important Beginning from iTest Release 5.2 onwards, the Keystore file property does not support JKS format, and any existing JKS key storages have to be converted to PEM format.

How to convert JKS file to PEM file:

To convert a Java Key Store (JKS) file into a Privacy Enhanced Mail Certificate (PEM) file, use the following two tools:

1. keytool.exe	to import the keystore from JKS to PKCS12. This tool is supplied with Java.
2. openssl.exe	to convert the PCKS12 to PEM. This tool is supplied with OpenSSL.

Follow these steps:

1. Use the **keytool** to convert the JKS into PKCS:

   ```
   > keytool -importkeystore -srckeystore client.jks -destkeystore client.pkcs
   -srcstoretype JKS -deststoretype PKCS12
   ```

2. Use `openssl.exe` to convert the PKCS into PEM:

   ```
   > openssl pkcs12 -in client.pkcs -out client.pem
   ```

3. Run each of these programs and enter passwords for the key stores (when prompted).
4. Use `client.pem` for Secure Authentication in the REST session profile.

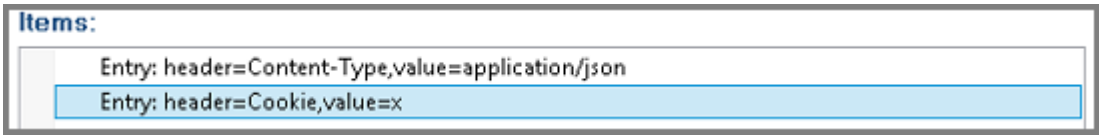
HTTP Headers

Headers that you specify here are added to each request URL.

To add a header field, uncheck the **Include inherited values** box and click **Add**  .

Header	Specify the field name of the header to send with each request
Value	Specify the value for the header.

You may select the HTTP Headers from a template list. Click  to display the available header templates and select the required template. The illustration below lists the HTTP Header templates available default:



See "HTTP Header Templates" below for creating custom templates.

HTTP Header Templates

Parameters that you specify here are stored as a separate list of headers in the session profile. You may select these header templates from a list, add to each (header list) request URL, and modify as needed after inserting.

To add a parameter field, uncheck the **Include inherited values** box and the Items column/box displays a list of existing Header templates. **Add**  and enter the parameters below.

Name	Specify the field name of the parameter to send with each request
Value	Specify the value for the header.

Proxy

Parameters that you specify here are used as Proxy settings for REST sessions. Any traffic routed through the proxy will be available for recording by the REST sessions.

To use Proxy setting, select the **Use Proxy** box and enter the parameters below.

Hostname	Specify the local host name to be used as proxy server.
Port	Specify the port through which the proxy server routes traffic.
Username	Specify the name of the user authorized to use the proxy settings.
Password	Specify the password associated with the user name.

URL Parameters

Parameters that you specify here are added to each request URL.

To add a parameter field, uncheck the **Include inherited values** box and click **Add**  .

Name	Specify the field name of the parameter to send with each request
Value	Specify the value for the header.

REST session action reference

The following actions are available in REST session steps in test cases, which describes using HTTP Methods for RESTful Services. Each entry describes the function of the action, and lists the return values of the primary HTTP methods (Safe and Idempotent Methods) in combination with the resource URIs. See also http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

Safe Methods	Conventions established considers methods that have action of only retrieval (**GET** and **HEAD** methods).as safe. These conventions allows user agents to represent other methods, such as **POST**, **PUT** and **DELETE**, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.
Idempotent Methods	A sequence/action that never has any side effects is defined as idempotent (provided that no concurrent operations are being executed on the same set of resources). The methods **GET**, **HEAD**, **PUT** and **DELETE**, **OPTIONS** and **TRACE** have no side effects, and are inherently idempotent.

POST

The POST method is used to create new resources and upload a file. When creating a new resource, POST action to the parent and the service takes care of associating the new resource with the parent and assigns an ID (new resource URI).

Action	POST - Create new resources
Returns	On successful creation, POST returns HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.: **Entire List:** HTTP 201 (Created), 'Location' header with link to /customers/{id} containing new ID. **Specific Item:** HTTP 404 (Not Found), 409 (Conflict) if resource already exists.
Method	POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.
Example	POST http://www.example.com/customers POST http://www.example.com/customers/12345/orders

GET

The GET method is used to read (or retrieve) a representation of a resource.

Action	GET- read (or retrieve) a representation of a resource
Returns	GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). **Entire List:** HTTP 200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists. **Specific Item:** HTTP 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. **Error:** HTTP 404 (NOT FOUND) or 400 (BAD REQUEST)
Method	GET (along with HEAD) requests when used to only read data this way, they are considered safe. Calling GET once has the same effect as calling it 10 times, or none at all. Additionally, GET (and HEAD) is idempotent, which means that making multiple identical requests ends up having the same result as a single request.
Example	GET http://www.example.com/customers/12345 GET http://www.example.com/customers/12345/orders GET http://www.example.com/buckets/sample

PUT

The PUT method is used to update (PUT to) a known resource URI with the request body containing the newly-updated representation of the original resource. PUT may also be used to create a resource where the resource ID is chosen by the client instead of by the server and upload a file. If a new resource is created, see "POST" on page 1195 for details.

CAUTION Use the PUT method for create purposes with caution.

Action	PUT - update (PUT to) a known resource URI with the request body containing the newly-updated representation of the original resource
Returns	On successful update, return 200 (or 204 if not returning any content in the body). **Note** If using PUT for create, return HTTP status 201 on successful creation. A body in the response is optional—providing one consumes more bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID. **Entire List:** HTTP 404 (Not Found), unless you want to update/replace every resource in the entire collection. **Specific Item:** HTTP 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
Method	PUT is not a safe operation as it modifies (or creates) state on the server, but it is idempotent. That is, creating or updating a resource using PUT multiple times does not change the resource state, it remains the same as the first PUT action.
Example	PUT http://www.example.com/customers/12345 PUT http://www.example.com/customers/12345/orders/98765 PUT http://www.example.com/buckets/secret_stuff

PATCH

The PATCH method is used to apply (PATCH) partial modifications to a resource, a known resource URI with the request body containing updates to the existing representation of the original resource. The set of changes is represented in a format called a PATCH document identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource, depending on the patch document type (whether it can logically modify a null resource) and permissions, etc. If a new resource is created, see "PUT" on page 1196 and "POST" on page 1195 for details.

CAUTION Use the PATCH method for update purposes with caution. See also https://tools.ietf.org/html/rfc5789

Action	PATCH - update (PATCH to) a known resource URI with the request body containing the updates to the original resource.
Returns	On successful update, return 200 (or 204 if not returning any content in the body). **Note** If using PATCH for create, the 204 response code is used if the response does not carry a message body (which a response with the 200 code would have). **Entire List:** HTTP 404 (Not Found), unless you want to update/replace every resource in the entire collection. **Specific Item:** HTTP 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
Method	The PATCH method affects the resource identified by the Request-URI, and it also MAY have side effects on other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH. With PATCH, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.
Example	PATCH /file.txt HTTP/1.1 Host: www.example.com Content-Type: application/example If-Match: "e0023aa4e" Content-Length: 100 [description of changes]

DELETE

The DELETE method is used to delete/remove a resource identified by a URI

Action	DELETE - use to delete/remove a resource identified by a URI
Returns	Successful deletion returns: • HTTP status 200 (OK) along with a response body, that is, representation of the deleted item (may take too much bandwidth), or a wrapped response • HTTP status 204 (NO CONTENT) with no response body. **Note** HTTP status 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses. **Entire List:** HTTP 404 (Not Found), unless you want to delete the whole collection—not often desirable. **Specific Item:** HTTP200 (OK). 404 (Not Found), if ID not found or invalid.
Method	The DELETE operation is considered idempotent. Deleting action removes a resource and repeatedly calling DELETE on that resource results with the response: as the resource does not exist. Calling DELETE on a resource a second time returns a HTTP 404 (NOT FOUND) since it was already removed and cannot be found.
Example	DELETE http://www.example.com/customers/12345 DELETE http://www.example.com/customers/12345/orders DELETE http://www.example.com/bucket/sample

OPTIONS

The OPTIONS method represents allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Action	OPTIONS - determines the options and/or requirements associated with a resource, or the capabilities of a server. • If the Request-URI is an asterisk (*), the OPTIONS request applies to the server in general rather than to a specific resource. • If the Request-URI is *not* an asterisk (*), the OPTIONS request applies only to the options that are available when communicating with that resource.
Returns	HTTP200 response includes any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow).
Method	Is inherently idempotent.as it has no side effects.
Example	OPTIONS /users/me returns: 200 OK Allow: HEAD,GET,PUT,DELETE,OPTIONS

HEAD

The HEAD method may be used for obtaining meta information about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The HEAD method is identical to "GET" (see "GET" on page 1195) except that the server does not return a message-body in the response. The meta information contained in the HTTP headers in response to a HEAD request is identical to the information sent in response to a GET request.

TRACE

The TRACE method is used to invoke a remote application-layer loop- back of the request message.

Action	TRACE - allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.
Returns	A valid request displays the entire request message in the entity-body of a 200 (OK) response. with a Content-Type of "message/http".
Method	Is inherently idempotent.as it has no side effects.
Example	--

CHAPTER 49

RFT Launcher (IBM Rational Functional Tester) Sessions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Launching Rational Functional Test sessions from iTest

You can use Spirent iTest to launch tests in IBM Rational Functional Test. The session window for Spirent iTest RFT sessions has been designed in partnership with IBM to enable you to launch Rational Functional Tester scripts from Spirent iTest. This integration is certified by IBM as *Ready for IBM Rational Software*.

Spirent iTest captures your commands to execute scripts so you can automate the execution and the analysis of responses.

Creating a test case that includes RFT sessions

Step 1 Define a test script

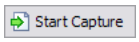
Use Functional Tester to define a test script in the normal way. Save it into the script folder.

Step 2 Configure the session

Configure a Spirent iTest RFT session profile as described in “Session profile property settings for RFT sessions” on page 938.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again (as described in “About property settings” on page 72).

Step 3 Start an interactive session

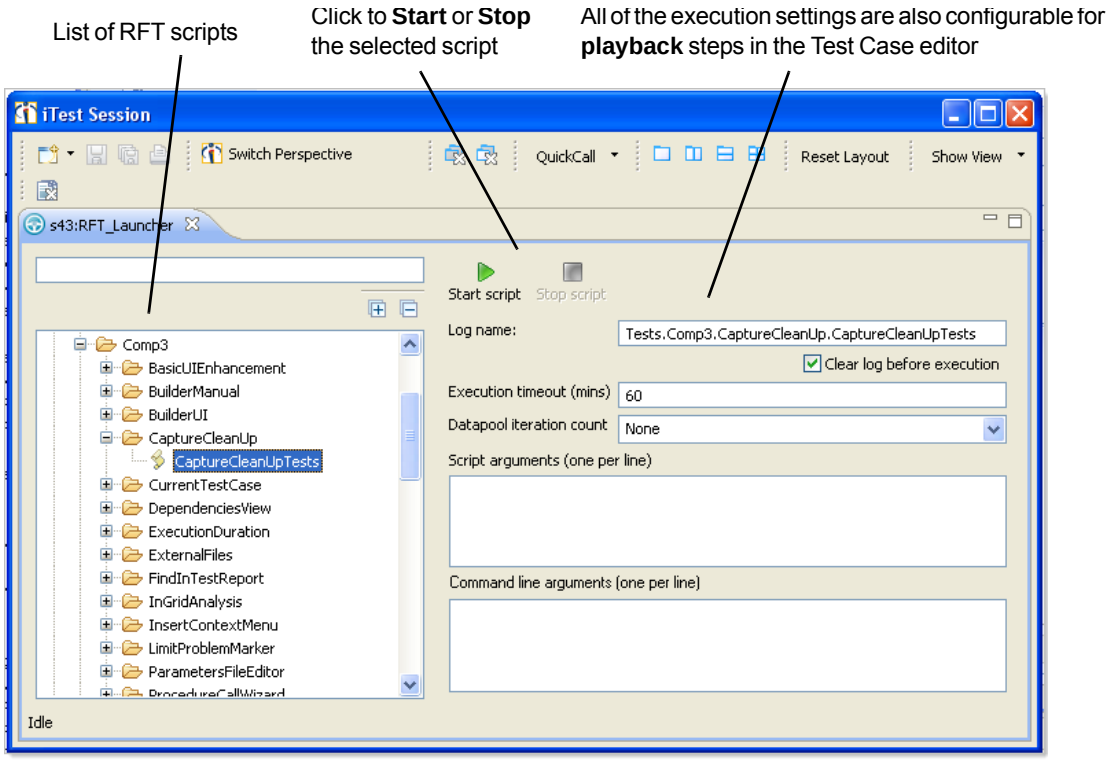
1. Click 

to begin the direct-to-test process of saving the interactive session as a test case.
2. Now launch the RFT session in Spirent iTest, either interactively (as shown in the following example), or as part of an automated test. To launch a script from an interactive session:

a Select the script.

b Specify the log file name and any arguments as described in the table.

c Click 



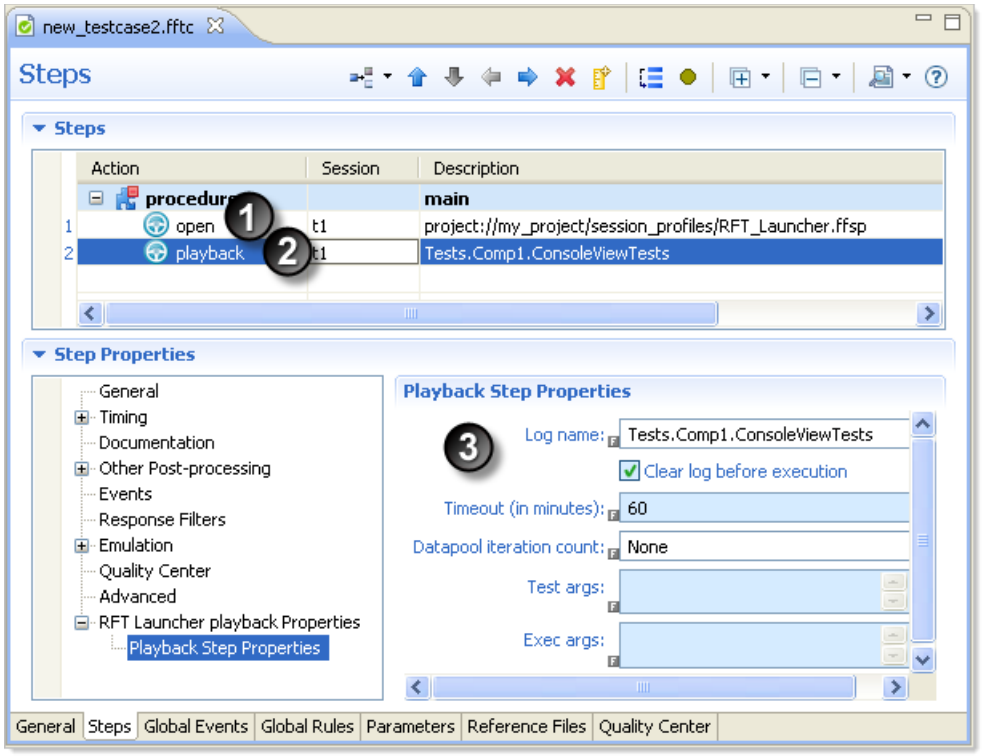
Log name	Specify the name of the log file to generate.
Clear log before execution	Check the box to overwrite the log when execution starts. If you uncheck the box, then a wizard asks you to specify a new name to use for the log file.
Execution timeout	Specify the maximum time in minutes that the script is allowed to execute. For a Spirent iTest test case, if test duration exceeds the specified limit, then the test case is aborted and marked **Fail**. Default: 60
Datapool iteration count	If the script is associated with a datapool, specify the number of times that the script should execute. Default: None
Script arguments	Optional. Specify each argument for the script. Specify each argument on a separate line.
Extra arguments	Optional. Specify arguments that should go in the execution command line of the script, for example, classpath. Specify each argument on a separate line. **Note** Arguments that are supported at the command line are listed in IBM's reference guide. Search for **[Class rational_ft]**

Spirent iTest captures the real-time data and displays it in the Console view. The RFT script produces a log file that you can analyze when the test concludes.

3 Execute as many scripts as needed in the interactive RFT session. When execution finishes, click Finish Capture to save the captured steps into a test case. (The **Add Test Case** wizard opens and help you through the process. For details on saving captured steps to test cases, see “Creating a test case by capturing interactive sessions” on page 125.)

Step 4 Edit the new test case

The **Test Case** editor opens the new test case. Let’s look at the steps in the test case and the properties of the selected **playback** step.



1 **The open step for the session** You opened a session and Spirent iTest auto-assigns a **session ID** of **t1** to the resulting **open** step.

2 **The playback step** Spirent iTest generated this **playback** step when you started the script by clicking . The **playback** action executes the script whose path is specified in the **Description** cell.

Important Because the response to a **playback** step is the text content of the log file, you can apply a Spirent iTest analysis rule directly to the step.

---

3 **Step properties for the playback step** In addition to the name of the log file for the script, Spirent iTest captured all of the settings that you configured when you started the script. You can view (and edit) all settings in the **Step Properties** section for the step.

Session profile property settings for RFT sessions

RFT

Functional Tester install dir	Specify the path to the installation directory for IBM Functional Tester.
Functional Tester project	Specify the path to the directory that holds the Functional Tester scripts. The contents of this directory is displayed as a structured list when you start the Spirent iTest RFT session.

Advanced

Log directory	Optional. Specify the path to the execution log file. Default: blank
Working directory	Optional. Specify the path to the folder where the RFT player is launched. Default: blank

Rational Quality Manager sessions in iTest

The iTest integration with Rational Quality Manager is certified by IBM as *Ready for IBM Rational Software*.

Overview: How you can use iTest and RQM

You can create a Rational Quality Manager (RQM) test case that executes a iTest test case. When you choose to run an automated test from within RQM, RQM sends commands to the devices in the test lab, causing the iTest test case to run and the results to be displayed in your browser.

iTest integrates with RQM through one or more **iTest RQM Adapters**. Each adapter is an agent that listens for execution requests from RQM. When it receives a request, it executes the specified iTest test case on the computer that hosts iTest. The adapter is installed as part of iTest installation on each computer that runs iTest.

The iTest-RQM integration supports the following activities from Rational Quality Manager:

- Run iTest test cases on remote lab computers
- View all available iTest test cases when adding iTest test cases to an RQM script
- For each script of type **iTest** (that is, any script that executes a iTest test case), you can specify the iTest topology and parameter file to use during execution and the execution tags to apply to the test report.
- View the current execution status of iTest test cases that have been run, are scheduled to run, or are currently running

Server Rename

The iTest RQM Adapter fully supports the **Server Rename** capability in the Rational solution for CLM. To use server rename:

1. Follow the instructions for server rename in the Rational documentation:

   https://jazz.net/help-dev/clm/topic/com.ibm.jazz.install.doc/topics/c_redeploy_server.html
2. Once the server has been renamed, update the server URL in the adapter startup script.
3. Restart the adapter.

Typical workflow for integrating RQM and iTest

1. Prepare a test case in iTest in the normal way (typically by capturing an interactive session).
2. Save the test case and all supporting resources as an ITAR.
3. Log in to the RQM server.
4. In the **Construction** menu, select **Create Test Case**. Specify a name for the new test case and any other settings that you typically set.
5. In the **Construction** menu, select **Create Test Script**. Specify a name for the new script.

   Specify a **Type** of **iTest**.

   Check **Select test resources that are local to a test machine**.

   Specify the following iTest resources:

Test Case	Required. Specify the iTest test case to run.
Testbed	Optional. Specify a topology or testbed to use for execution. This topology or testbed overrides the resource specified for the test case.
Parameter File	Optional. Specify a parameter file to use for execution. This parameter file overrides the resource specified for the test case.
Tags	Optional. Specify text that should be associated with the test report. • Tags are used only if test reports are saved to an external database. • Each report is tagged with the text. • You can use the value to search for test reports in the database. Specify tag settings using **name=value** pairs that are separated by commas. For example, **Environment=Linux, Type=Stress** **Note** Tags are not used with the iTestRT **--comparereports** option. **Pre-defined tags:** • **Group:** The text appears in the **Group** column on the Review Test Reports activity page and on the Test Reports view. • **Subgroup:** The text appears in the **Subgroup** column on the Review Test Reports activity page and on the Test Reports view.

6. On the test case tab (named with the new test case name), select **Test Scripts** from the **Table of Contents**.
7. Click + to add a test script.
8. Start a iTest RQM adapter as described in “Launching a iTest RQM adapter” on page 941.
9. In RQM, in the **Execution** menu, select **Adapter Console**.

   The **Adapter Console** tab displays the list of adapters (both active and inactive).

   Click **Refresh** to update the page to display all adapters. Verify that the adapter appears (the **Machine Name** is the text that you specified for the **instanceID** argument when you ran iTestRT) with a green circle indicating that it is active.

10 On the test case tab, select the **Machine** (RQM adapter) that should run the iTest test case.

Click execute .

The test runs on the iTest computer and the adapter returns links to test reports to the RQM server.

11 Click **Close and show results**. The **Result Details** section displays a link to the iTest test report.

Launching a iTest RQM adapter

You start adapters from iTestRT. Here are the RQM adapter arguments for iTestRT:

--licenseServer	Required. See “Checking out a runtime license” on page 710.
--rqm.adapter.projectArea *area*	Project area for RQM server
--rqm.adapter	Required. Run iTestRT as RQM adapter
--rqm.adapter.repository *URL*	RQM server URL To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
--rqm.adapter.pollInterval *interval*	Polling interval in seconds
--rqm.adapter.retryCount *count*	Maximum number of re-connection attempts
--rqm.adapter.user *userName*	Username for RQM server
--rqm.adapter.password *password*	Password for RQM server
--rqm.adapter.instanceID *ID*	Instance ID for RQM server. This text appears in the **Adapter Console** tab as the **Machine Name.**
--rqm.adapter.config *URI*	Configuration file location (can be used to set all other adapter properties).

Example iTestRT command to launch a iTest RQM adapter

```
itestrt.bat --rqm.adapter --rqm.adapter.repository
https://10.11.12.13:9443/jazz --rqm.adapter.user <userName>
--rqm.adapter.password <password> --rqm.adapter.projectArea iTestRT_adapter
--rqm.adapter.instanceID MyAdapter --rqm.adapter.pollInterval 5
--rqm.adapter.retryCount 3
```

CHAPTER 65

Script Library Support sessions

Script Library Support: Executing your existing external procedure libraries from Spirent iTest

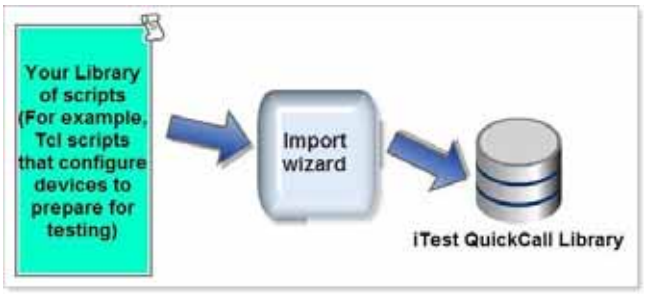
Important

The Spirent Script Library Support sessions are ***obsolete and is no longer supported*.**

This chapter is provided only to support existing implementations.

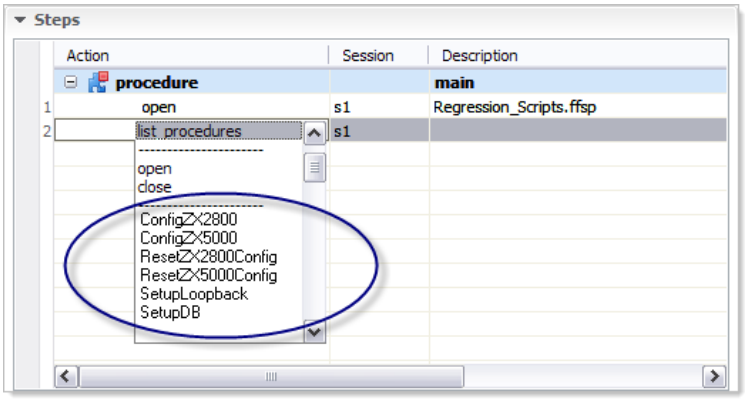
The Script Library Support session type enables your iTest test cases to invoke functions and procedures that are defined in external script libraries (your existing “home-grown” procedure libraries).

To enable a iTest test case to directly call an external procedure, you use a wizard to import one or more libraries. The wizard converts the call to each selected procedure from the libraries into a QuickCall in a QuickCall library. The wizard then creates a Script Library Support session profile that refers to the library.



Later, while developing a test case in the Test Case editor, when you open a Script Library Support session, each procedure is available in the **Action** cell as a QuickCall. (For more information on QuickCalls, see Chapter 10, “QuickCalls: Defining and using a library of custom actions”.)

Arguments appear in the **Description** cell (you can specify arguments in the **Command** property).



When the test case runs, the QuickCall step invokes the external procedure to run *in its native environment*. The procedure returns data to the iTest environment. iTest converts the raw responses (for example, Tcl arrays, lists, tables, and so on) into structured data. As a result, the responses are easier for a person to read in the Response view and analysis rules are easier to define.

Creating the library in iTest: Importing an external library of procedures

The import process generates two files:

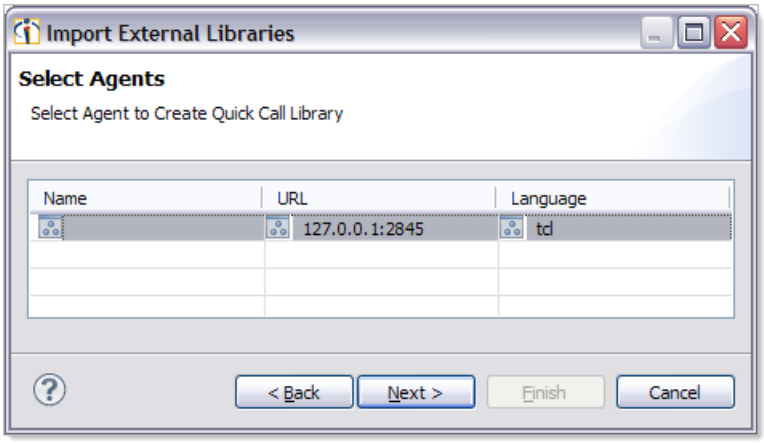
- A QuickCall library — a test case where the iTest QuickCalls that call the external procedures are defined
- A Script Library Support session profile — a session profile that refers to the QuickCall library

Later, when you work on a test case step in the Script Library Support session, each of the external procedures appears in the **Actions** list as a QuickCall.

Follow these steps to import an external library of procedures:

1. Ensure that your system administrator has configured the agents (as described in “Prepare the agents that convert the external procedures” on page 1211).
2. Start the **Import** wizard: click **File > Import**. On the **Select** page, select **iTest > External Procedure Libraries**.

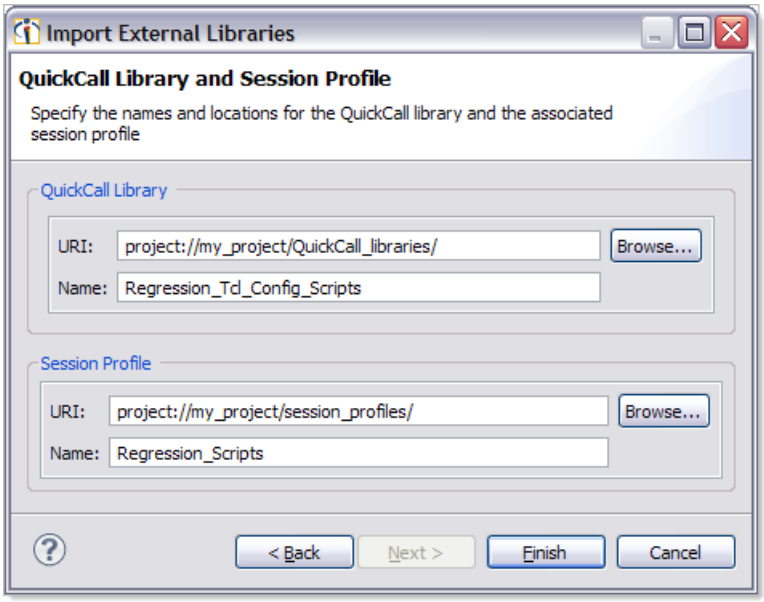
3 On the **Select Agents** page, select the agent for the language of the scripts that you will import (**Tcl** in the example).



4 On the **Select Procedures** page, specify which procedures to import — either individual procedures or entire libraries.

Note When the procedures are converted to QuickCalls, characters in the procedure names that are not allowed in iTest procedure names are replaced with the _ underscore character.

5 Specify the locations and names for the QuickCall library and the session profile for the Script Library Support session that should refer to it.



6 Click **Finish**.

Session profile property settings for Script Library Support sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

About Agents

Software *agents* find external scripting libraries and convert them into QuickCall libraries. Agents also provide the interface between iTest and the native script environment during execution. An agent supports a particular scripting language (for example an agent might support Tcl scripts).

Script Library Support

Agent language	Scripting language that the agent supports when searching for scripts and converting them into QuickCalls.
Render response as plain text	Check the box to render the response for all test case steps as plain text (not structured text). Uncheck to use the setting in the **Response Rendering** step property for each step. Default: Unchecked **Note** If you select **plain text**, then the **Response Rendering > Render response as plain text** setting is ignored for all steps in the test case. See the description for the **invokeProc** action in “Command reference” on page 1214.

Prepare the agents that convert the external procedures

Option A: Run the Agents on the iTest Team Essentials/ Agent Registry server

When you run agents on iTE, the agents are a shared resource available to anyone running iTest. In contrast, an agent running on a host that is running iTest is available only to the local instance of iTest.

1. Use the Startup command to connect an agent to the server. Specifying port 9000 to connect to the registry server and specify the IP address or hostname of a remote host.

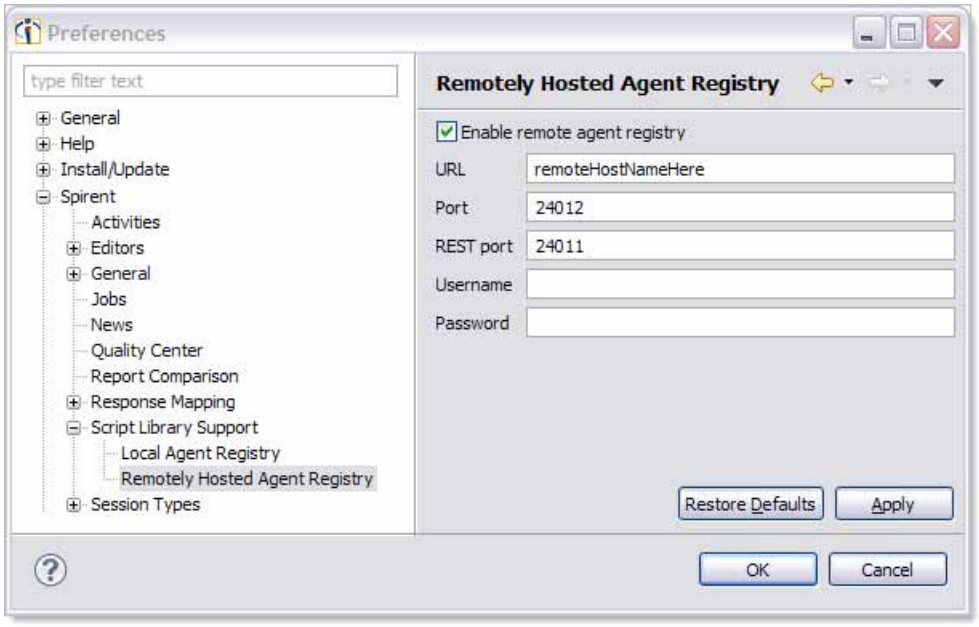
   ```
   Startup.sh/cmd -l <library Name> -P 9000 -h <remoteHost>
   ```

2. When the connection is made, the remote host prints an initiating packet response.

Note The only way to distinguish between agents is by the language the agent is using, so use the `-l` option when configuring multiple agents.

3 In iTest, on the **Preferences** page (**Window > Preferences**), navigate to **Spirent > QuickAccess >**.

Check **Enable remote agent registry** and set the property values as shown in the example:



4 Restart iTest. The agent should appear in the Script Library Support Agent List view as a valid agent.

Note Remember that multiple people can work with a server at the same time, so more agents may appear than you expect.

Option B: Run the agents on the host

Linux

Install the Tcl interpreter by running one of the following commands:

- To install the standard Tcl libraries (like cmdline) that are used by the agent:

  ```
  sudo yum install tcllib
  ```

- To install tbcload:

  ```
  sudo yum install tbcload
  ```

Microsoft Windows

Step 1 Configure system environment variables

TCL_AGENT_PATH	The folder that iTest Tcl agent lives in. For example, C:\Program Files\Spirent Communications\iTest 4.2\ScriptLibrarySupport
LAUNCHER	Full path to the Tcl interpreter. For example, C:\Tcl\bin\tclsh.exe
TCLLIBPATH	Full path to the Tcl/lib folder.

Step 2 Configure iTest preference settings

1. Start iTest.
2. In iTest, click **Window > Preferences** to open the **Preferences** page.
3. Navigate to **Spirent > Script Library Support > Local Agent Registry**. Change the local agent **Port** to **17776**. (Default value is **-1**)
4. Restart iTest.

Step 3 Run the TCL agent

1. With iTest running, at the command prompt, navigate to the Tcl agent directory and then run the startup.cmd command. For example:

   `>cd C:\Program Files\Spirent Communications\iTest 4.2\ScriptLibrarySupport`

   `>startup.cmd -P 17776 -l` ***<libraryName>***

2. Now verify that the agent is running. In iTest, the **Connect to Agents** view displays the list of active agents. Click **Window > Show View > Other > AgentConnector > Connect to Agents**.

Step 4 Prepare your script libraries

Copy the script libraries into the Tcl/lib folder (typically `C:\Tcl\lib`).

Script Library Support Agent List view

Software agents find external scripting libraries and convert them into QuickCall libraries. Script agents also provide the interface during execution. An agent supports a particular scripting language.

To help you when importing a library of scripts, the Script Library Support Agent List view (**Window > Show View > Other > Spirent > Script Agent**) lists all currently running agents.

Advanced users only: Script Library Support command set

This topic describes all Script Library Support commands that you can submit from iTest.

Important This section is for advanced troubleshooting only—you do not typically work with any of the following actions.

For a step in an Script Library Support session, select one of the listed actions in the **Action** cell.

Command reference

invokeExpr	Invoke the expression specified by the string in the **Description** cell. **Note** See “Step property setting for invokeExpr, invokeFile, and invokeProc” on page 1214
invokeFile	Execute the file that is specified by the Windows file path. • If the path includes spaces, then enclose the path in quotes • If the path includes backslash characters, then either disable backslash substitution or enclose the path in quotes or curly braces { } **Note** See “Step property setting for invokeExpr, invokeFile, and invokeProc” on page 1214
invokeProc	Invoke the procedure specified by the string in the **Description** cell. *namespace procedure [argument argument ...]* Use a **return** action to return the response. **Note** See “Step property setting for invokeExpr, invokeFile, and invokeProc” on page 1214
list_procedures	List all procedures that are defined in the library.

Step property setting for invokeExpr, invokeFile, and invokeProc

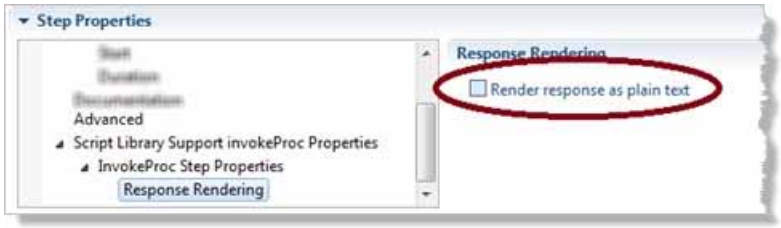
You can set the following step property for **invokeExpr**, **invokeFile**, and **invokeProc** actions:

Script Library Support <actionName> Properties >

<actionName> Step Properties >

Response Rendering >

Render response as plain text



- Check the box to render the response for the step as plain text.
- Uncheck to render the response as structured text.

Default: Unchecked

If the property setting for the session is set to **plain text**, then this setting is ignored. See “Session profile property settings for Script Library Support sessions” on page 1211

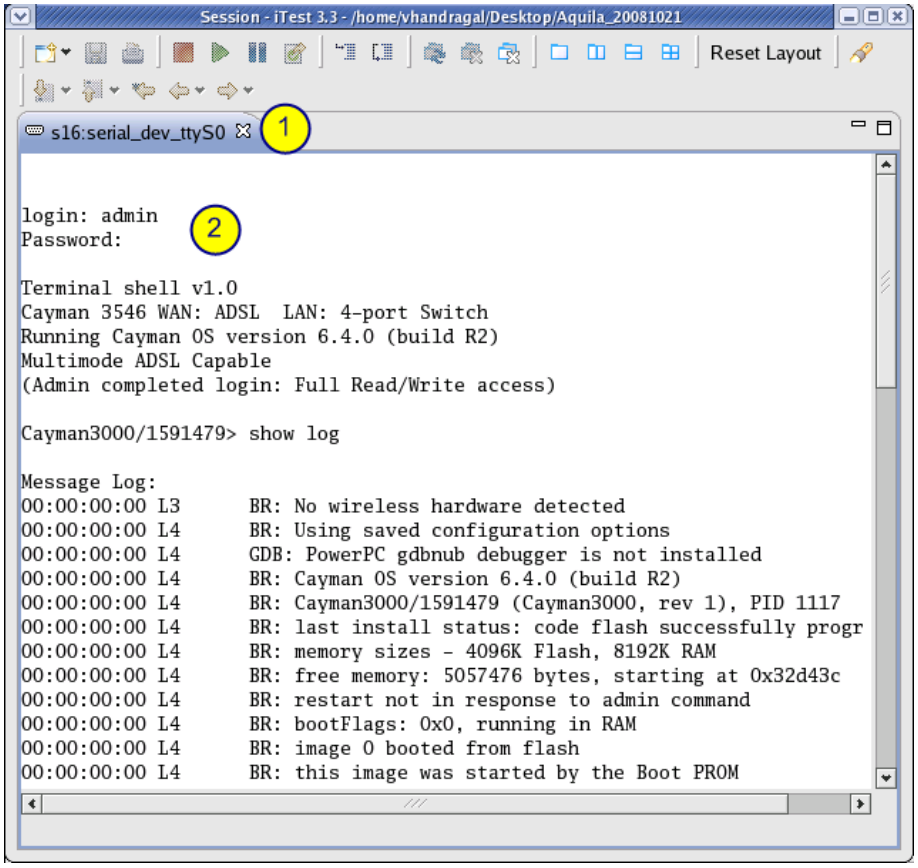
CHAPTER 51

Serial Sessions

Serial session window

For Serial sessions, the computer running iTest communicates directly over a serial port connection with the device under test.

For each open session, the Serial Port session window displays your commands and the device's responses. You can think of the session window as a terminal client — a terminal that iTest is monitoring and capturing.



The session window looks just like a client terminal session. When you type a command and press Enter, the editor displays both the command text that you submitted and the device's response. (The command/response pair (plus some session information) is called a ***captured item*** and is listed in the Capture view.)

1 The tab for the session window displays the session type icon and the name of the session profile that was used to start the session as the **Session ID** (**serial_dev_ttyS0** in this example).

2 This example session shows the login process, a **show log** command, and the device's response.

When you close a session (for example, by issuing an **exit** command), iTest captures a **close** Action and then dims the session window.

Tips for interactive sessions

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

Session profile property settings for Serial sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Serial

Port	Specify the serial port for the session, for example, COM1 on Microsoft Windows or /dev/ttyS0 on Linux. For Windows, iTest identifies the available serial ports and displays them in a drop-down list. For other platforms, you must type the port name. Default: [blank]
Baud rate	Specify the connection speed. Default: 9600

Serial Port > Advanced

Parity	Specify the Parity bit in each character. Default: None
Handshake	Specify the hardware handshake signal for the session. Default: None
Data bits	Specify the number of Data bits in each character. Default: 8
Stop bits	Specify the number of Stop bits sent at the end of every character. Default: 1
Send newline character (\n) when session connects	By default, iTest sends a newline character as soon as a Serial session connects. Uncheck the box to ***not*** send any characters upon connection. Default: checked

Serial Port > High Availability

For details on implementing tests for HA devices, see Chapter 38, “Testing High-Availability (HA) Devices”.

High Availability	Check the box to enable HA operation. (The default setting, unchecked, specifies normal, non-HA operation.)
Additional connections	Specify the IP address and port pair for each redundant node (nodes other than the master node.). This information is used only by the **open** step for a session. The values in the list represent nodes 1, 2, 3, ... n. Use the following format, one node per line: **<IP_or_hostname>:<portnumber>** **Important**: Be sure not to enter the values for node 0 — the master node — those values are specified by the **IP Address** and **Port** properties.

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Keyboard

Backspace	Specify the code that the device interprets as the backspace character. Default: Ctrl-H
Enter	Specify the code that the device interprets as the Enter character. Default: \r\n

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Style

Note For session profiles that will support TL1 devices, see “Configuring sessions and test case steps for TL1 devices” on page 1215.

Style	Specify the expected format of the responses to commands. **Normal**: The responses will be text (either structured or not structured). **TL1**: The responses will use TL1 formatting. Default: Normal

Terminal > Prompts

For an overview on how prompts work, see “Overview: Prompts in iTest” on page 463.

For instructions on using the properties in this group to define prompts, see “Editing prompt definitions” on page 467.

For related prompt properties, see “Terminal > Replay > Step Defaults > Completion” on page 953.

Name	The name helps you to remember the type of prompt, for example, **LoginPrompt**. Default: [various]
Content	Specify the exact text of the prompt. **Note**: All prompt definitions are case-insensitive and leading and trailing whitespace is trimmed from any prompt text before iTest attempts to determine whether response text is a prompt. If you use regular expressions in the **Content** value, then set the **Type** property to **Regex**. If the prompt includes a space character or any whitespace in the body of the text, be sure to set the **Type** property to **Wildcard**. Default: [none]

Type	Specify the kind of prompt. **Normal**: Interpret the text in the Content field as the case-insensitive text that you expect for the prompt. **Wildcard**: Disregard any characters that appear in the location of the * character in the text specified for the **Content** property. The most common application for the **Wildcard** setting is to allow for leading or trailing numeric or UserID characters in the prompt (for example Device02>, Device03>, and so on). If you set **Type=Wildcard**, then only the * wildcard character is allowed within the **Content** string (and no other wildcard characters like ?). To use other wildcard characters in the **Content** string, you must use **Type=Regex**. **Regex**: Interpret the text specified for the **Content** property as a regular expression. Default: [none]
Is more prompt **More next command** **More quit command**	The **-- more --** prompt is a common method for allowing command line users to view one screen (page) at a time. Many devices use the space character as the command to move to the next page (and often, the letter **q** to exit the display of the response). To enable your automated test cases to page through data that is displayed one page at a time, iTest can automatically “press the space bar” as often as is required to get to the end of the response. As a result, the device's response to the command becomes a single uninterrupted flow of text that does not include the More text. If the prompt is a page-control prompt (for example - - more - -, then: Select the **Is More prompt** checkbox. In the **More next command** text box, specify the command characters (typically a space character) that cause the next page to appear. By default, a space character appears in the box. In the **More quit command** text box, specify the command that exits the More display and returns to the command line prompt. By default, a **q** character appears in the box. Specify a value for **Terminal** > **Replay** > **Step Defaults** > **More**.

Terminal > Replay > Step Defaults > Command

Send interval between each character	Some devices require a delay between characters in order to receive commands correctly. If required, specify the interval of time to wait before sending each character in a command. Default: 0 milliseconds

Terminal > Replay > Step Defaults > Terminator

Line terminator	The settings enable you to configure non-standard terminator settings for a session's behavior during replay. For example, special commands like **show ?** do not require the user to press Enter on some devices, but other devices do require the user to press Enter. If you specify **Custom**, then you must specify the terminator text for the **Custom line terminator** property. Typically, you do not need to make any changes to this setting. Default: The default setting (**\n**) is newline.
Custom line terminator	See **Line terminator**. Default: [blank]

Terminal > Replay > Step Defaults > Response

Treat LF as CRLF	Check the box to cause iTest to change “LF” (linefeed) characters that are returned by the session into “CRLF” (carriage return / linefeed). This is helpful for sessions that send line endings as “LF” only (for example, Python shell). Default: unchecked
Filename to write response to	Specify the URI of a file to write the responses into. You can use field replacements in the text of the URI to allow the test case to set the filename at runtime. For example, **file:subdirectory_name/[param file_to_create]** The full text of the response is written to the file regardless of the **Number of lines to keep** setting. Default: <none> See the following associated properties: **Append response to file** **Response header** **Number of lines to keep** **Write echo to file** **Write prompt to file**
Response header	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: You may want to specify a text string that should appear before each block of response text. For example: **+-+-+-+-+- Next Response Starts Here -+-+-+-+-+** Default: <none>

Number of lines to keep	For very long responses, you might not want to keep all of the response text (as displayed in the Response view for the selected step while working in the Test Report editor or in the Test Case editor). Specify the maximum number of lines to keep for any single response. Specify **0** (zero) to keep all lines. **Note** If you specify a URI in the **Filename to write response to** property, then all lines in the response are written to the file, regardless of this setting. Default: 10,000
Append response to file	If you save responses to a file by specifying a value for the Filename to write response to property, then: Check the box to append each new response to the file specified in the **Filename to write response to** property. Uncheck the box to replace the text of the file specified in the **Filename to write response to** property with the most recent response. As a result, the file will hold only the last response in the session. See the **Filename to write response to** and **Response header** properties. Default: Checked
Write echo to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: Check the box to include any echoed characters in the saved response. Default: Checked

Write prompt to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: • Check the box to include the last line of the response in the saved response (in command-line applications, this is typically the prompt after the response). • Uncheck the box to not save the last line of the response to the file (the prompt at the beginning of the response where the command was typed is still saved). Default: checked

Note

Options **Write echo to file** and **Write prompt to file** do not work in capture mode. These options are available for **Replay** mode only.

The example below shows how the commands/responses are echoed in these scenarios.

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options *are selected*:

```
prompt>command
response text
etc
etc
prompt>
```

Write echo to file

- When **Write echo to file** is *not selected*

```
response text
etc
etc
prompt>
```

- When **Write prompt to file** is *not selected*

```
prompt>command
response text
etc
etc
```

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options are *not selected*

```
response text
etc
etc
```

This is because the **Terminal > Replay** options are used to replay the captured steps. That is, replay the steps captured via test case execution or replayed from the Capture View (“Working in the Capture view” on page 103).

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal > Replay > Step Defaults > More

Pages to fetch	For responses that are longer than be displayed on a single screen, devices often provide a page-control prompt that enables you to view one screen of text at a time (for example **- - more - -**). Specify the number of pages to fetch when the **more** prompt appears (zero means get all pages). If the setting is non-zero, then iTest retrieves that number of pages and then terminates the output from the session's response by sending the command specified for the **More: Quit Command** property. Default: 100
Device does not remove more prompt. Remove more prompt from response.	Some devices do not remove the text of the more prompt from the text of the response. (For these devices, you will see the more prompt remain at the bottom of the page even after you press the continuation character — typically the spacebar.) Check the box to eliminate the more prompt text from the response that is saved by iTest. Default: Unchecked
Use BELL character to detect end of more pages	Some devices do not remove the more prompt from the screen even after you even after you press the continuation or quit character. Such devices often use the audible bell to alert the user that they have reached the end of the response. Check the box to cause iTest to use the bell as its indicator that the response is complete. Default: Checked

Terminal > Replay > Step Defaults > Completion

You use **Completion** settings to define when the execution of a step should be considered complete. The determination of when a step is complete is protocol-specific. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- For some steps, you might have defined analysis logic to examine the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

For CLI protocols, you can specify any of several conditions to define when the step is complete, for example, the existence of certain text in the response or the time elapsed after sending the command. The default setting of the **Completion criteria** property is that the step is complete when:

a The session channel is idle for the time specified by the **Idle channel interval** property

and

b The last line of the response matches one of the prompt definitions specified for the session profile or device.

Idle channel interval	This setting helps in cases where you do not know what response to expect and can use a specified idle time (for example, 100 milliseconds) or when you expect no response whatsoever, for example, when talking to a terminal server. Default: 100
Wait for first character before starting idle	Some devices do not respond to a typed command immediately. This setting enables you to ignore the idle time after the last character of the command is echoed and the first character of the actual response is returned. This way, the delay is not misinterpreted as idle channel time for the purpose of determining completion. Default: True
Completion criteria	**Prompt matches AND device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property and last line of the response matches one of the prompt definitions specified for the session profile. **Prompt matches OR device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property **or** the last line of the response matches one of the prompt definitions specified for the session profile. The following processing order occurs: The step is completed once one of the defined prompts is received. If none of the defined prompts is received or no prompt is defined, then the system waits for the specified **Idle channel interval** time (during which the device sends no response data) and then completes the step. **Device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property. **Completion time has expired:** The step is complete when the time specified by the **Completion time** property has elapsed. If you specify **Completion time has expired**, then the **Idle channel interval** property setting is ignored. **TL1 End of Message**: For session profiles that will support TL1 devices, see “Configuring sessions and test case steps for TL1 devices” on page 1215 Default:**Prompt matches AND device has not sent data during the Idle channel interval**
Completion time	Specify the time interval that must elapse for the step to be complete. To apply this setting during execution, the **Completion citeria** property must be set to **Completion time has expired**.

Where to find prompt	Specify where the prompt in a response normally appears. **Last line** **Last non-empty line** The **Any line** setting is a special case that you can use to detect a change in state for an ongoing response. For example, you can detect a port's connection status based on whether the first character of a ping response is **u** or **s**. Be sure to use wildcard characters as needed in the **Content** property. Default: Last line
Command to send when a step is cancelled	Specify the characters to send when the user cancels step execution. Default: \03 (Ctrl-C)
Capture only the last screen of response text	Use this property when you expect a very large response, but the only data of interest appears at the end of the response text. Check the box to cause iTest to save only the last screen of response text. Default: Unchecked
Unknown Prompts During Automated Execution For an overview on how iTest recognizes prompts, see “Overview: Prompts in iTest” on page 463. For instructions on defining prompts, see “Editing prompt definitions” on page 467.	
Expected maximum Idle channel interval	The time to wait for a prompt during automated execution. When this time is reached, iTest displays a **Learn this prompt** link in the status bar. • If the user clicks the link, then iTest opens the **Learn Prompt** dialog box to enable you to add the prompt definition. • If the user chooses not to click the link and the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues. Default: 5
Extra wait before alerting user	Additional time to wait for a prompt during automated execution after the **Expected maximum Idle channel interval** time has been exceeded. When (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed, then: • A countdown timer in the status bar starts to count down the **Time for user to respond** time period. • iTest displays a **Keep waiting** link in the status bar. • If the user clicks the **Keep waiting** link, then iTest waits for an additional (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed period. • If the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues Default: 15
Time for user to respond	Specify the amount of time in seconds to wait for the user to respond once the status bar displays the **Waiting for prompt** timer. Default: 30

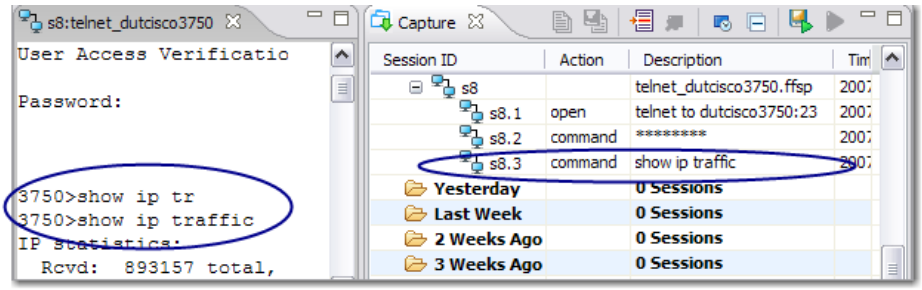
Terminal > Replay > Step Defaults > High Availability

See “Testing HA devices: Detailed instructions” on page 822.

Terminal > Capture

Perform capture cleanup	When you perform manual testing in CLI sessions, you frequently use meta-characters like backspace and up- and down-arrows to correct your typing. In a Capture report, such commands can be difficult to read and understand. If you check **Perform capture cleanup**, then iTest "cleans up" any keyboard shortcuts and removes meta-characters from the captured commands so that the resulting command text appears as if you typed it fully and correctly. See the **Discard command completion steps** property**.** Default: Checked A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, even though we actually typed **show ip tr**<tab>. iTest discarded the intermediate **show ip tr**<tab> form of the command.
Mask unechoed commands	Check **Mask unechoed commands** so that, before creating a Capture report, iTest masks all **Command** property text for which no echo was returned. Check the box to automatically mask passwords. Default: Checked
Remove echo from response	Check **Remove echo from response** to indicate that the device echoes characters typed at the command line. In this case, iTest ignores echoed characters so that the command text is not added to the actual response text. Default: Checked
Remove prompt from response	Check **Remove prompt from response** to save only the response from the session and not the prompt text. We recommend that you do not disable this setting except in rare circumstances. Default: Checked
Use prompts from the session for cleanup	Check the box to use the prompt definitions specified in the session profile document when cleaning up commands. Default: Checked
Discard command completion steps	To ensure that captured commands in the Capture view are easy to understand, iTest, by default, deletes the intermediate command completion text that was submitted while forming a command. A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, and discarded the intermediate **show ip tr**<tab> form of the command. See the **Perform capture cleanup** property. Default: Checked
Learn prompts	If the box is checked, then, when you close a session and iTest has detected a new prompt, the Update Session Profile wizard starts. You can specify particular prompts in the **Terminal > Prompts** properties. Default: Checked
Learn command completion characters	Learn the character code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can). If the box is checked, then, when you close a session and iTest has detected a new command completion character, t the Update Session Profile wizard starts. The default completion character is tab. To specify particular characters, configure the **Terminal > Capture > Command Completion** property. Default: Checked

Learn break characters	Learn the character code that the device interprets as a break (so you can manually cancel an executing step). The learned break characters are added to the **Command break characters** property. If the box is checked, then, when you close a session and iTest has detected a new break character, the Update Session Profile wizard starts. Default: Checked. The default break character is Ctrl-C. **Note** To specify particular break characters manually, configure the **Command break characters** property (in **Terminal > Capture > Break**).
Remove line containing more prompt	Check the box so that, when capturing responses, iTest deletes the lines that include the more prompt. Default: Checked



Terminal > Capture > Response

Number of lines to keep	Specify the number of lines in the response to keep in the Capture report. Default: 10,000

Terminal > Capture > Command Completion

Specify the code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can).

Default: tab (\t **)**

To determine the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters.

Limitations of the Record feature:

- For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”.
- Function keys are not recorded.

Command completion requires ENTER key	Most devices respond immediately with command completion when they encounter one of the characters specified for the **Command completion characters** property. Set this property to TRUE, if the device, to perform command completion, requires that you press ENTER after typing one of the characters specified for the **Command completion characters** property. Default: Unchecked

Terminal > Capture > Break

Command break characters	Specify the character code that the device interprets as a break (so you can manually cancel an executing step). Default: Ctrl-C To add the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters. **Limitations of the Record feature:** • For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”. • Function keys are not recorded.

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

CHAPTER 52

SNMP Sessions

SNMP session window

iTest supports the Simple Network Management Protocol: SNMPv1, SNMPv2C, and SNMPv3.

The SNMP session window is a MIB browser that displays the tree for the specified MIB on the specified device, the variable definitions from the MIB file, and the value of a selected variable.

You can replay captured actions by dropping them from the Capture view into the SNMP session window.

Most devices support SNMP for monitoring and configuration. Because you typically use one software tool for CLI and a different tool for SNMP, it is difficult to automate SNMP tests. As a result, over time, a device's CLI command interface can get out of synch with the SNMP interface. iTest makes it easy to compare the operation of a device's CLI interface and the SNMP interface in a single test case. For example, a test case can send traffic and then compare the SNMP responses to the CLI responses.

You will use the SNMP session window to get and set MIB variables.

WARNING MIB parser sometimes cannot handle duplicate names very well (if you load only DOCS-QOS3-MIB-110210.txt then there is no problem). This issue will be addressed in the future release.

Important: Loading MIB definitions

- Sessions may load MIB files from various locations. This may result in odd behavior if the files have conflicting definitions.
- By default, to ensure faster execution, iTest loads a MIB file into the MIB cache only one time during the lifetime of the current instance of iTest. iTest loads MIB files when a session starts (either the default MIB definitions or the MIB files that are specified in the session profile). Once a MIB file has been loaded, however, iTest does not reload it (even if the session starts again). This means that:
  - Changes that you make to a MIB file after a session has run are not loaded when the session starts again.
  - MIB errors that were displayed the first time a MIB file was loaded (for example, when a session started) are not displayed again even if the session starts again.

  To avoid this issue, enable the **Reload MIB files when session starts** session property, as described in "SNMP MIB Browser > MIBs" on page 972.

Traps

iTest receives traps in two ways:

- At any time, whether a iTest SNMP session is running or not, iTest receive traps on the ports specified in the iTest preferences (**Window > Preferences**.**Spirent > Session Types > SNMP**).
- When a iTest SNMP session is running, it communicates with one SNMP agent. The session can bind to a port to listen for SNMP traps from the agent (the port is specified for the **Trap Port** properties in the session profile). While the session receives traps only from the agent, the SNMP Traps view shows traps received from any agent.

See “Configuring trap settings” on page 964.

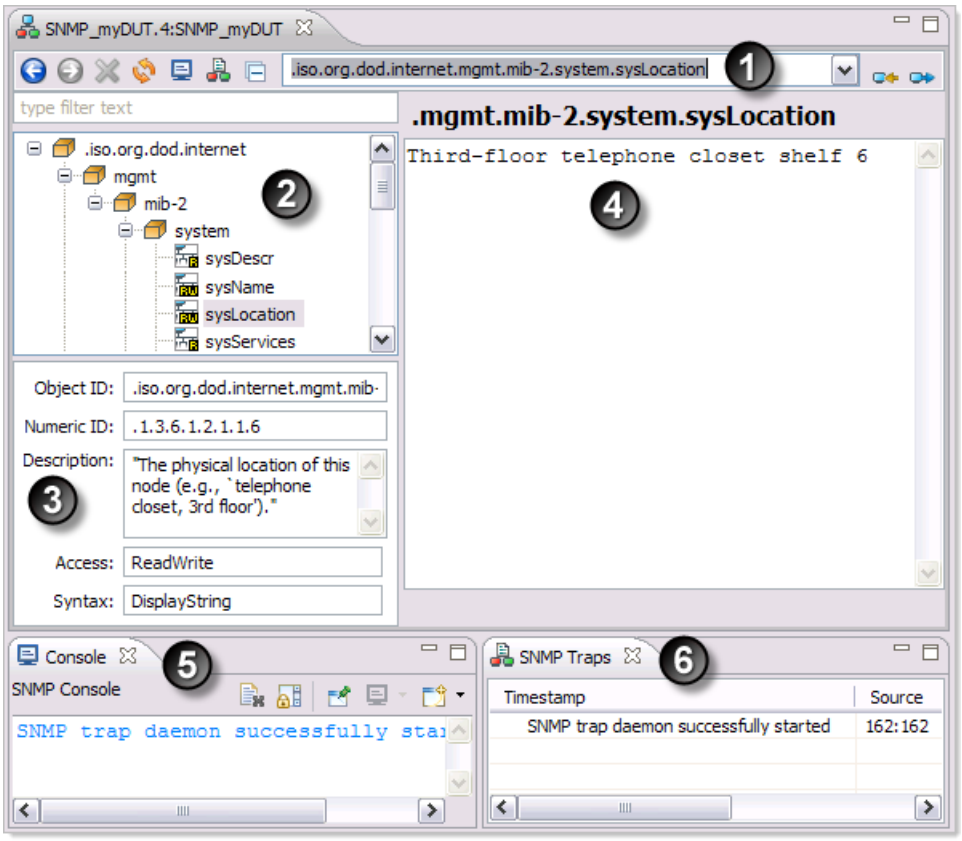
Specifying MIB definitions to load

Follow the directions on setting the **MIBs** property in “Loading your proprietary MIB files into iTest” on page 978.

The SNMP session window

1 Toolbar and OID text box. When you select a variable in the tree 2, its OID appears in the box.

Tip To display the scalar value of a variable, either select the variable in the tree or type the OID in the OID text box with “.0” at the end and then press **Enter** or click **Get**.



❷ Navigate the MIB tree and select a variable to view its properties ❸ and to view its value in the value editing pane ❹.

R: read-only, **RW:** read-write, key: locked

Tip While browsing for a variable, you can limit what appears in the tree by typing filter text in the filter text box. Only variables that include the filter text are listed. Wildcard characters ***** and **?** are supported.

Setting and **get**ting values:

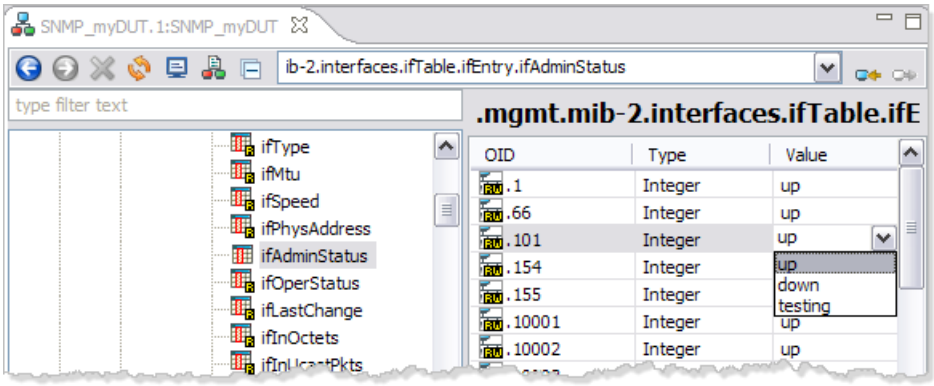
Get and set operations are asynchronous and can be interrupted by navigating elsewhere or by clicking **Cancel**.

To **get** a value: Double-click the variable or select it and press ENTER. The value appears in the value editing pane ❹.

To **set** a value:

- **Option A:** Select the variable. In the OID text box ❶, type a space after the OID. Type the new value and then press Enter (or click **Set**).
- **Option B:** Select the variable. In the value editing pane ❹, modify the value or type the new value and then click **Set** (for table entries, you can press Enter).

Note For table variables and variables that you 'walk' to: If the variable definition includes a syntax map, then the value pane provides a drop-down list of the "friendly" values defined in the map. In this example for the item named **.101**, the friendly values **up**, **down**, and **testing** are specified in the syntax map and correspond to the integer values **1**, **2**, and **3** that are actually used by the **set** action.



❹ To replay captured actions, drop them from the Capture view into the value editing pane.

❺ The SNMP Console view displays a text log of all traps received by iTest. The same information appears in table format in the SNMP Traps view.

❻ The SNMP Traps view displays a table of information on all traps received by iTest from any agent.

Getting values

Use any of the following methods:

- Double-click the variable in the MIB tree. iTest performs the appropriate action (**Get**, **GetTable**, or **Walk**), and then displays the results in the value editing pane.
- Select the variable in the MIB tree and then press Enter.
- Type or paste the variable's OID into the OID text box and then press **Enter** or click **Get** .

Setting scalar values

1. First **get** the value. The value appears in the value editing pane.
2. Select the value to make it editable, type or paste the new value, and then press **Enter** or click **Set** .

Setting table values and values that you "walk to"

1. First **get** the values.

   Double-click a container node to Walk multiple scalar values. The values appear in a grid in the value editing pane. The headings are **OID**, **Type**, and **Value**.

   Double-click a table node to use **GetTable** to get table values. The values appear in table format in the value editing pane.

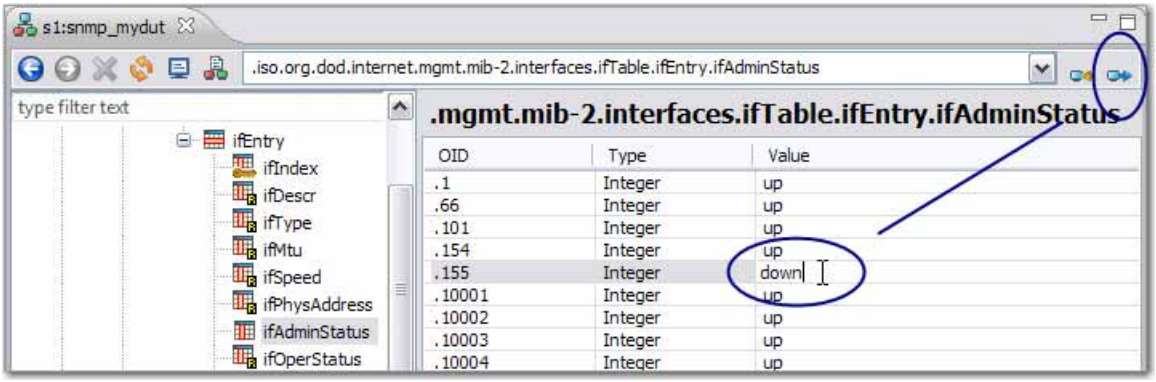
   Walk: Select the **Value** cell in the grid that corresponds to a read/write node ( ).

   Tables: The selected cell is editable if the column is read/write( ).

Note Read-only and Read/Write icons appear next to variables in the value editing pane. Read-only variables are also dimmed to indicate that you cannot edit the value.

2. Select the value to make it editable, type or select the new value, and then press **Enter** or click **Set** .

Note For table variables and variables that you 'walk' to: If the variable definition includes a syntax map, then the value pane provides a drop-down list of the "friendly" values defined in the map. In this example for the item named **.101**, the friendly values **up**, **down**, and **testing** are specified in the syntax map and correspond to the integer values **1**, **2**, and **3** that are actually used for the **set** action.



SNMP actions that are captured for replay

The following SNMP actions and their responses are captured while you browse and edit an SNMP variable.

To save the responses, save the captured items as a Capture report.

To replay captured items, drop selected items onto the SNMP editor.

Get	Returns the OID and Value of a single MIB variable. iTest uses the Get PDU to get values for scalar variables. To get a value, double-click a scalar variable  or  } in the MIB tree. The **get** Action and the returned Value are captured.
GetTable	Returns all **OIDs**, **Types**, and **Values** of variables in the table. To use **GetTable**, double-click a MIB table  in the MIB tree. iTest uses a Get PDU with multiple OIDs to read the values for each row.
Set	Sets the value of a single MIB variable. 1. To **set** a value, first **get** the value. 2. Select the value to make it editable, type the new value, and then press Enter (alternatively, click **Set** in the toolbar). iTest uses a **Set** PDU to set the value. In the example that appears below this table, we selected a read-write variable in a table, changed its value, and then **set** the value.
Walk	Returns all of the data in a MIB under a specified root. For SNMPv2c or SNMPv3, when use **GETBULK** is enabled, iTest uses the **GetBulk** PDU to get values for container nodes For SNMPv1 or when use **GETBULK** is disabled, iTest uses the Get PDU for each OID. To use **walk** in an interactive session, double-click a container node 

SNMP session window (MIB browser) toolbar

 	**Get** the previous / next value in the navigation history.
	Cancel the current operation.
	Refresh the MIB tree and the values in the value editing pane.
	Open the SNMP Console view
	Open the SNMP Traps view
	Collapse all open containers in the MIB tree.
	**Get** the value of the variable listed in the OID text box.
	**Set.** To set a value, select the value to make it editable, type the new value, and then click **Set**.

SNMP session window (MIB browser): Icons in the MIB tree

	Container node
 	Scalar MIB. R: read-only RW: read-write
	Table
 	Table column. R: read-only
	Table row
	Key column

Note During SNMP session replay, you can move the cursor and browse the MIB tree, but cannot affect the replay activities in any way. Pause replay to perform offline operations.

Configuring trap settings

iTest receives traps in two ways:

- At any time, whether a iTest SNMP session is running or not, iTest receive traps on the ports specified in the iTest preferences (**Window > Preferences**.**Spirent > Session Types > SNMP**). See “Setting preferences for monitoring SNMP traps” on page 982.
- When a iTest SNMP session is running, it receives traps only on the port specified for the **Trap Port** properties in the session profile. See “SNMP MIB Browser > Traps” on page 973 and “SNMP MIB Browser > Step Defaults > Traps” on page 978.

Creating SNMP test case steps

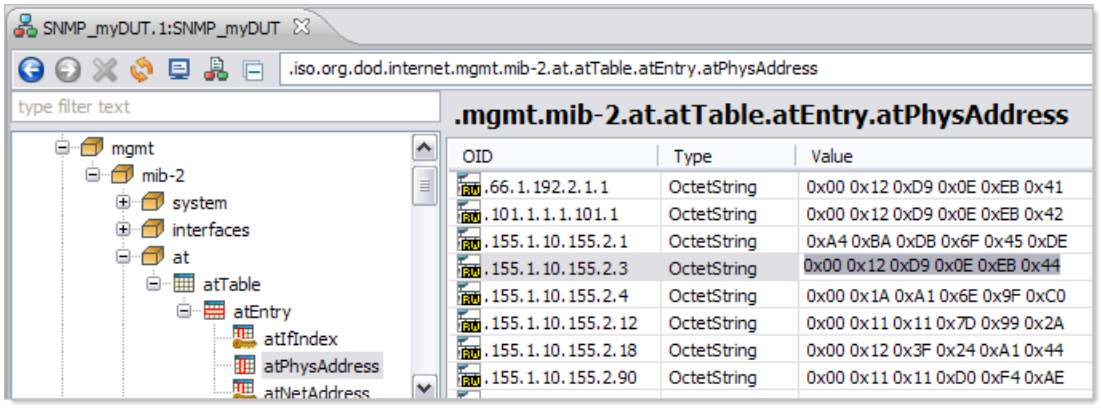
As with most session types, the easiest way to create an SNMP test case is to manually execute the steps that you expect to include in the test case and then save the captured session as a procedure in a test case.

Because SNMP data is structured, it is automatically parsed without requiring you to create a Response map.

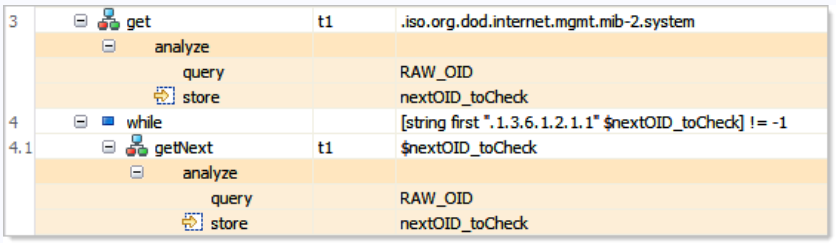
SNMP action types that are captured during interactive SNMP sessions

The following SNMP actions and their responses are captured while you browse and edit an SNMP MIB.

get	Returns the OID and Value of a single MIB variable. iTest uses the Get PDU to get values for scalar MIBs. The response contains the printable value of the specified MIB variable. For octet strings, the result is displayed as a string of 2-digit hex values for each byte, separated by colons (typical MAC address format). For strings, non-printable characters (except for CR/LF) are converted into printable versions using the standard iTest field replacement syntax (for example, **[char CTRL-C]** or **[char \t]**).
getTable	Returns all OIDs, Types, and Values of variables in the table. To use GetTable in an interactive session, double-click a MIB table in the MIB tree. iTest uses a Get PDU with multiple OIDs to read the values for each row.
set	Sets the value of a single MIB variable. iTest uses a Set PDU to set the value. In the example that appears below the table, we selected a read-write variable in a table, and can change and then set its value.
walk	Returns all of the data in a MIB under a specified root. For SNMPv2c or SNMPv3, when use GETBULK is enabled, iTest uses the GetBulk PDU to get values for container nodes For SNMPv1 or when use GETBULK is disabled, iTest uses the Get PDU for each OID.



SNMP action types that you can perform in test case steps

get	Returns the OID and Value of a single MIB variable. iTest uses the Get PDU to get values for scalar MIBs. The response contains the printable value of the specified MIB variable. For octet strings, the result is displayed as a string of 2-digit hex values for each byte, separated by colons (typical MAC address format). For strings, non-printable characters (except for CR/LF) are converted into printable versions using the standard iTest field replacement syntax (for example, **[char CTRL-C]** or **[char \t]**).
getNext	Returns the value of the single MIB variable that follows the specified OID. The structured data includes the OID value and Spirent iTest generates queries for OID and RAW_OID. **Tip** To use **getNext** in a loop for returning multiple values, the device's agent must implement a variable (the "next" variable) for loop control. If you intend to get values for all variables in a MIB, use **walk** instead. **Example getNext in a While loop** In this example, we obtain the OID and use it to control a **while** loop around the **getNext**: 
getTable	Returns all OIDs, Types, and Values of variables in the table. iTest uses a Get PDU with multiple OIDs to read the values for each row.
set	Sets the value of a single MIB variable. iTest uses a Set PDU to set the value. **Note** You have the option to configure iTest to execute a **get** action before executing any **set** action. See the session profile setting described in "SNMP MIB Browser > Step Defaults > Set" on page 977.
walk	Returns all of the data in a MIB under a specified root. Returns the following statistics: • Number of nodes visited • Deepest node level visited • Number of errors happened (error codes from API) For SNMPv2c or SNMPv3, when **Use GETBULK** is enabled, iTest uses the GetBulk PDU to get values for container nodes. For SNMPv1 or when **Use GETBULK** is disabled, iTest uses the Get PDU for each OID. **Tip** To enable tests to exit infinite loops caused by self-referencing OIDs, consider setting the **Stop on cycle** property for the session. See "SNMP MIB Browser > Step Defaults > Walk" on page 977.

listTraps	• If a port is specified for the **SNMP port** property in the session associated with the step, then **listTraps** lists all traps that have been received on the port since executing the first SNMP step (or previous **listTraps** calls, depending on the **Clear Traps After List** property setting as described in a moment). Traps on the ports specified in preferences are not included in the response. • If no port is specified for the **SNMP port** property, then traps on the ports specified in preferences are included in the response. • If no port is specified for the **SNMP port** property and no port is specified in preferences, then all traps are included in the response. **listTraps** steps do not send commands to the device. Depending on the **Clear Traps After List** property setting in the session profile, iTest either leaves the queue unchanged or clears it after **listTraps** steps. The response body contains an XML document listing the following elements for each trap: • Timestamp • Trap objects See “SNMP Traps view” on page 982
waitForTrap	• If a port is specified for the **SNMP port** property in the session associated with the step, then **waitForTrap** causes execution to wait until a trap is received for the port. Traps on the ports specified in preferences are not considered. • If no port is specified for the **SNMP port** property, then **waitForTrap** causes execution to wait until a trap is received for one of the ports specified in preferences • If no port is specified for the **SNMP port** property and no port is specified in preferences, then **waitForTrap** causes execution to wait until a trap is received for any port. **waitForTrap** has two modes: • If the step specifies an OID in the **Command** property, then **waitForTrap** waits for the specified trap (or any trap with the specified prefix). • If the step does not specify an OID in the **Command** property, then **waitForTrap** waits for any trap. Depending on the Use received traps for wait property setting in the session profile, the step executes as follows: If Use received traps for wait is selected (default), then iTest first checks the receive queue. If a trap is in the queue (either any trap or a trap for the specified MIB, as appropriate), execution proceeds. If the queue is empty, then execution proceeds upon receipt of a trap (either any trap or a trap for the specified MIB, as appropriate). If Use received traps for wait is not selected, then iTest ignores the queue and awaits a trap (either any trap or a trap for the specified MIB, as appropriate). If non-matching traps are received, they are added to the queue. See the description of the **Use received traps for wait** property. Depending on the **Clear after listing** property setting, iTest leaves the queue unchanged or clears it after **waitForTrap** steps. **Response** The response contains an XML document listing the following elements for the trap that ended the wait: • Timestamp • Trap objects The response will always include one trap unless the step times out or is canceled. If the step times out or is canceled, then the response is an XML document listing no traps. See “SNMP Traps view” on page 982.

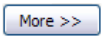
Configuring trap settings

See “Configuring trap settings” on page 964.

Session profile property settings for SNMP sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

SNMP MIB Browser

IP address	Specify the IP address or hostname of the agent that you want to interact with in the SNMP session window
SNMP port	Specify the port for the agent. Default: **161**
SNMP version	Specify the SNMP protocol version of the agent. Default: **V2c**
Read community	**Note** This property appears only for SNMPv1 and SNMPv2c. Specify the **Read community** string (the community to use for requests to the agent). Default: **public**
Write community	**Note** This property appears only for SNMPv1 and SNMPv2c. Specify the **Write community** string (the community to use for SNMP SET requests). Default: [none]
OctetString representation	**Note** This property appears only for SNMPv1 and SNMPv2c. Select one of these SNMP compatibility options to specify how OctetStrings are represented in the responses. • Hex: value in hex mode “0x00 0xBA...” • Human-readable: non-printable characters are replaced with dots • Auto Hex: switch to Hex if the text has non-printable characters • iTest 4.3 compatibility mode Default: **iTest 4.3 compatibility mode**
User name	**Note** This property appears only for SNMPv3. Specify the user name.
Authentication password	**Note** This property appears only for SNMPv3. This property is disabled if the **No authentication, no privacy** option is selected for the **Security level** property in the **Authentication (SNMPv3 only)** property group. Specify the authentication password.

SNMP MIB Browser > Authentication (SNMPv3 only)

Note These settings apply only for SNMPv3.

V3 Context	Specify a context name (per RFC 2275) for the SNMPv3 request. Default: <empty string>
Security level	Specify the security level. Default: **No authentication, no privacy**
Authentication algorithm	Specify the authentication algorithm. Default: **MD5** **Note** The **Authentication password** is specified on the **SNMP MIB Browser** page.

Privacy	Specify the privacy password. Default: [none]
	Specify the encryption standard. Default: **DES**

SNMP MIB Browser > Aliases

Optional but recommended: To make it easier to select MIBs from lists in SNMP sessions, you can specify an alias for your proprietary set of MIB variables. For example, when you specify the alias name **ACME** to replace **iso.org.dod.internet.private.enterprises.acme**, the OIDs for the MIB variables in the session window's drop-down lists become much easier to read and select.

Example

When you specify an alias of **ACME**, then

```
ACME::IMAGE-MIB
```

appears in the list instead of the full OID prefix and MIB name:

```
iso.org.dod.internet.private.enterprises.acme.IMAGE-MIB
```

◆ To add an alias for a proprietary set of MIBs

1. Check **Include inherited values** to enable you to specify additional lists of proprietary MIBs. (For a discussion on inheriting MIB set aliases from reference session profiles, see “Property values: Inheriting settings” on page 62.)
2. Check **Include additional values from list.**
3. Click **Add**  to add an alias. The new alias appears in the list. Specify the following properties for the new alias:

Name	Specify a name for the alias. In the example, the **Name** is **ACME**. Default **Names** and **Contents**: MIB-2 / .iso.org.dod.internet.mgmt.mib-2 SNMPv2 / .iso.org.dod.internet.snmpV2
Content	Type or paste the OID prefix that you want to alias. In the example, the **Content** is: `iso.org.dod.internet.private.enterprises.acme`

4. During sessions, the sets of MIBs are searched in the listed order. Move any alias up or down in the list by selecting it and using the arrow buttons.

SNMP MIB Browser > MIBs

MIBs folder	By default, iTest loads MIB definitions from the default MIBs folder in the **resources** project: **project://resources/SNMP/Mibs/** The default paths are: Linux: ~/iTest/ workspace/resources/SNMP/Mibs Windows: C:\Documents and Settings\<user_name>\My Documents\iTest_<version>\resources\SNMP\Mibs For instructions on specifying additional or proprietary MIB definitions to load, see “Loading your proprietary MIB files into iTest” on page 978.
Strict MIB parsing	If unchecked, iTest loads all valid objects in the MIB file even if some objects in the file cannot be resolved. Check the box to cause iTest to perform strict MIB parsing: Before loading the MIB file, ensure that every SNMP MIB object in the file is fully resolved. Default: unchecked
Show duplicates in MIB tree	Check the box to display all MIBs that have the same OID. For example, if **.iso.org** and **.iso.dod** have the same OID of **.1.3**, they will both appear in the MIB tree (in random order). If unchecked, only one of the duplicate MIBs (selected randomly) will appear in the tree. Default: unchecked
Reload MIB files when session starts	By default, to ensure faster execution, iTest loads a MIB file into the MIB cache only one time during the lifetime of the current instance of iTest. iTest loads MIB files when a session starts (either the default MIB definitions or the MIB files that are specified in the session profile). Once a MIB file has been loaded, however, iTest does not reload it (even if the session starts again). This means that: • Changes that you make to a MIB file after a session has run are not loaded when the session starts again. • MIB errors that were displayed the first time a MIB file was loaded (for example, when a session started) are not displayed again even if the session starts again. Check the box to load all MIB files in the directory specified for **MIBs folder** each time the session starts. Default: unchecked

SNMP MIB Browser > Traps

Port	Specify the port where traps should be received by the **waitForTrap** and **listTraps** actions. The setting serves two functions: • Adds this port for listening (if the port is not configured in preferences) • Acts as a filter for incoming traps **Note** 1. See “Port binding” on page 973. 2. If no port is specified, then the session captures all traps from the ports specified in preferences, as noted in “iTest receives traps in two ways:” on page 973.
NIC address	Specify the IP address of the NIC where traps should be received. **Tip** To determine the IP address of the NICs on a computer, use the `ipconfig/all` command, **Note** When the receiving computer has multiple NICs: If you do not specify an IP address, then iTest listens to one of the NICs at random (on the specified **Port**).
Engine ID	This setting applies only for SNMPv3 Specify the **snmpEngineID**.
Override existing settings for same trap port bound when the session starts	Select to override the properties of specific trap port. When selected, SNMP trap port could rebind to new properties for every execution of SNMP session and allow reuse of trap port for different properties (e.g., different SNMP versions). When not selected, iTest does not allow binding different properties to the same trap port unless restarted. For example, when running tests and waiting for SNMP traps for version 2 and 3 at random on the same port, the SNMP sessions display error when switching between the two sessions. ■ When running an SNMPv3 test case after a successful SNMPv2c test case, an error may display as follows: SNMP trap daemon failed to bind to port N: a daemon is already running on the port with settings (V3=false) ■ Similarly, when running an SNMPv2c test case after a successful SNMPv3 test case: SNMP trap daemon failed to bind to port N: a daemon is already running on the port with settings (V3=true, ...)

Port binding

- Ports that you specify in the **SNMP MIB Browser > Traps > Port** property and in preferences remain bound while iTest is running.
- You can use multiple session profiles to bind to multiple ports on the same NIC at the same time.

iTest receives traps in two ways:

- At any time, whether a iTest SNMP session is running or not, iTest receive traps on the ports specified in the iTest preferences (**Window > Preferences**. **Spirent > Session Types > SNMP**).
- When a iTest SNMP session is running, it communicates with one SNMP agent. The session can bind to a port to listen for SNMP traps from the agent (the **Port** specified here).

The session receives traps only from the agent. The SNMP Traps view shows traps received from any agent.

The **waitForTrap** action has an empty **Command** property by default. **waitForTrap** waits for any trap received from the agent of the session on the **Port** specified here. If you specify the expected trap name in the **Command** property, the action waits for a trap from the agent with a name that starts with the specified text (the **Command** text acts as a prefix).

Privileges for SNMP traps

Executing the SNMP daemon on the Linux or Apple Macintosh operating system requires root privileges. Typically, however, you execute iTest as a regular user. As a result, you may not have privileges to listen to the default SNMP trap port 162. In this case, set a different trap port to listen on in either the SNMP session profile or in the iTest SNMP preferences settings. You must configure your SNMP agent to send traps on the new port.

SNMP MIB Browser > Step Defaults

General options	
Timeout (ms)	Specify the timeout in milliseconds for any single send/receive transaction. Default: **5000**
Retries	Specify the number of times to retry send/receive transactions. Default: **2**
Do not capture SNMP response	Check the box to conserve memory by not capturing the SNMP response. **Note** When you configure a step to save a response to a file (in the **Other Post-processing > Store Response** property group), this setting is ignored and all SNMP responses and statistics are always written to the file. Default: unchecked (False)
Do not capture SNMP statistics	Check the box to conserve memory by not capturing the SNMP statistics for the response. **Note** When you configure a step to save a response to a file (in the **Other Post-processing > Store Response** property group), this setting is ignored and all SNMP responses and statistics are always written to the file. Default: checked (True)
Get Bulk options	
Use GETBULK	When applicable for a get, use GETBULK to collect the response. Default: checked (True)
Max repetitions	Specifies the number of variables requested for each GETBULK request. Default: **50** Most agents make a best effort to fill their response with 50 variables, but may do fewer if they cannot fit 50 into a single PDU (which is dependent on the network configuration and other factors). In a situation like this, there is no harm in asking for 100 or more because iTest returns as many as possible. In other situations, you may need to lower the number to something the agent can handle.

SNMP MIB Browser > Step Defaults > GetTable

Maximum number of rows to fetch during automated execution

This setting applies only during automated execution (when replaying a step). Specify the maximum number of table rows to return for a **GetTable** action.

Default: 1000

iTest 3.1 compatibility mode

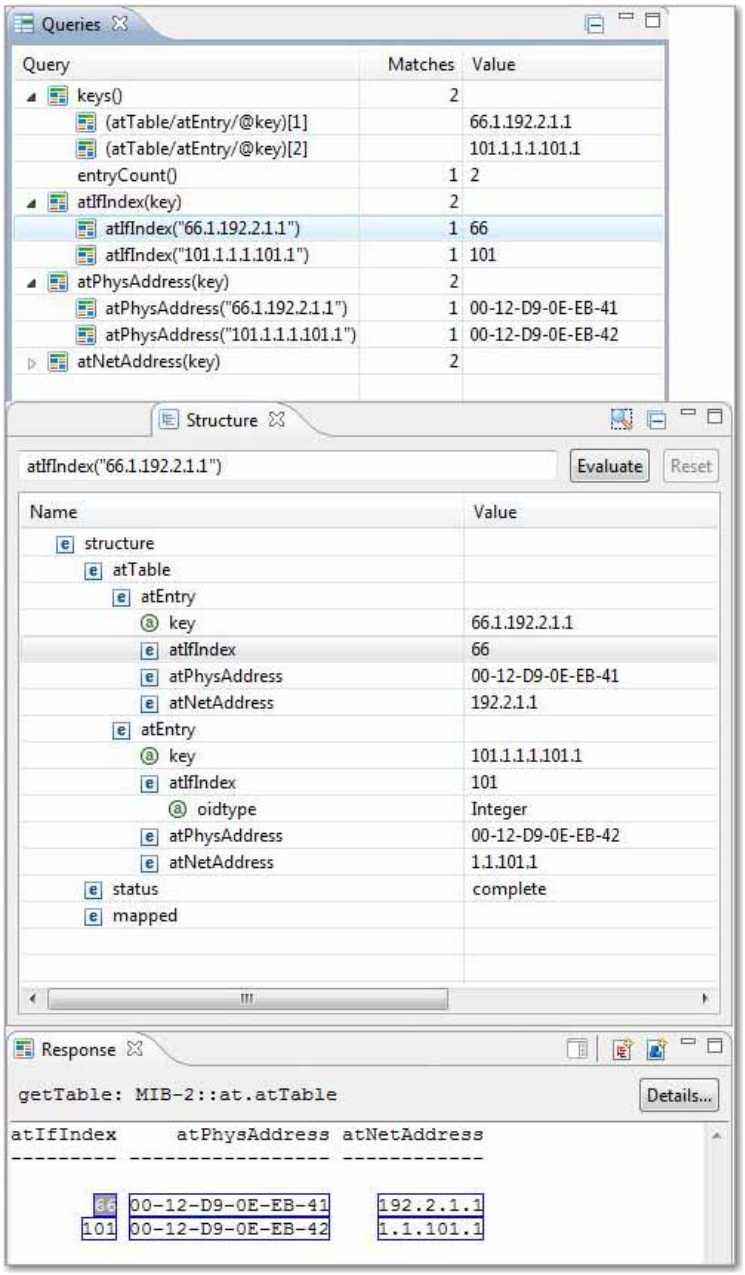
For iTest versions after 3.1, the format of the structured data for SNMP tables changed. This setting applies for SNMP **getTable** actions only.

iTest versions after 3.1 render the structured data with the row identifier (the part that gets suffixed to the OID) added as an attribute. The **key** attribute is added to the **entry** element (in the old format, the **oid** attribute was added to each field). The value for **key** is the same as the eliminated **oid** attribute and is up a level.

Each field gets a same-named query which takes **key** as its single argument. A **values** query will return all the keys.

Default: Unchecked

For example, **MIB-2::at.atTable** has a compound key:



SNMP MIB Browser > Step Defaults > Walk

Maximum number of items to fetch during automated execution	This setting applies only during automated execution (when replaying a step). Specify the maximum number of values to return for a **Walk** action. Default: 10,000
Stop on cycle	Enables tests to exit infinite loops caused by self-referencing OIDs and OIDs that incorrectly duplicate OIDs that appear earlier in the MIB. Default: unchecked
Trim ".0" from OID	During an interactive session, to view the scalar value of a variable, you can either • Click the variable in the MIB tree Use ".0" as the final characters of the OID in the OID text box and then press Enter or click **Get** . During interactive sessions, iTest performs the get operation and, by default, to make it easier to read the OID, strips the trailing ".0" text as displayed in the OID text box and in the command for the captured step. Adding the ".0" text explicitly ensures that iTest will use the full OID text in the command for the captured step. • Uncheck the box to display the trailing ".0" text in the OID text box and to capture it in the command for the step. • Check the box to strip the trailing ".0" text. Default: checked

SNMP MIB Browser > Step Defaults > Set

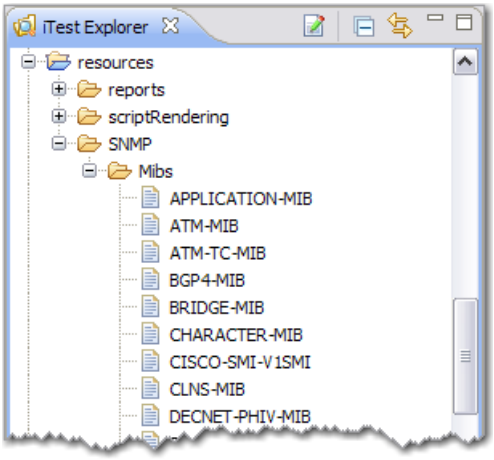
Execute a get action before executing set	Check the box to cause iTest to execute a **get** action before executing any **set** action.This option enables you to correctly set a variable whose type you do not know before execution — the type returned by the **get** action is used to perform the **set** action. Uncheck the box to execute only a **set** action for a **set** step. You specify the type using the **Value type** property. **Note** If you check the box, the **Value type** property is ignored. Default: unchecked
Value type	If you know the type of the value to set for any **set** action, then you can specify the type here. (This option is available only if you uncheck the **Execute a get action before executing set** check box.) **Note** It is typically best to set this property for an individual step in the Test Case editor. Default: [blank] (that is, no type is specified)

SNMP MIB Browser > Step Defaults > Traps

Clear the received traps list after listTraps action	Check the box (default) to cause all queued traps to be cleared after iTest executes a **listTraps** step. Uncheck the box to leave all queued traps in place. Default: checked
Use traps in the received traps list to trigger waitForTrap action	This setting affects the behavior of **waitForTrap** steps. Check the box (default) to cause **waitForTrap** steps to trigger for both traps that had been received before the **waitForTrap** step and for new traps. Uncheck the box to cause **waitForTrap** steps to ignore traps that were queued before the **waitForTrap** step and to trigger only for new traps. Default: checked
Remove matching trap from received traps list after waitForTrap action	Check the box (default) to delete (from the queued traps) any trap that triggers the **waitForTrap** step. Uncheck the box to leave all queued traps in place. Default: checked
Timeout for waitForTrap steps	Specify, in milliseconds, how long to wait for a trap before executing the next step. Default: 10,000 msec (10 seconds).

Loading your proprietary MIB files into iTest

iTest includes a large set of standard MIB files (including most MIBs specified by RFCs) in the resources/SNMP/Mibs folder. Often, you will need to use your organization's proprietary MIBs while testing devices. This topic shows you how to tell iTest to load additional MIBs.



Two options

There are two options for specifying which MIBs to load:

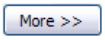
- **Specifying a different folder for MIBs:** You set a property that causes iTest to look in another folder. This option has the advantage that the MIB definitions are independent of the computer on which iTest is running (for example, when your test group uses a standard location for MIB files). Subfolders are not supported, so all of the individual MIB definition

files must appear in the specified folder. If you want to use the standard MIB files that iTest provides, then you must copy them to the folder.

- **Copying your MIBs into the default folder (Not recommended):** You copy your MIBs into the default folder (**resources/SNMP/Mibs**). Subfolders are not supported, so you copy the individual MIB definition files. This option is not recommended because it has the following disadvantage: If you discover problems with the custom MIBs, it might not be easy to locate and remove them.

Tip To ensure good performance, add only the MIBs that you expect to use for testing.

Specifying a shared folder for MIBs

1. Place all of your MIB definition files (uncompiled text format) into a particular directory. You must copy the individual files because subdirectories are not supported.
2. Open the device's session profile and click 

to view the **Session Properties** section.
3. In the tree, click **SNMP MIB Browser > MIBs**.
4. In the **MIBs folder** text box, type or paste the URI of the directory. For example, **myhost:///mymibs or filename:///c/mymibs**. (Remember, subdirectories are not supported.)
5. Click **Save**.
6. Exit and restart iTest.
7. The standard MIB definitions provided by Spirent appear in the **Mibs** directory. If you point to a new location and want to continue to use the Spirent default MIBs, then you must copy the default MIBs from **Mibs** to the new location.

Copying your MIBs into the default folder (Not Recommended)

1. Copy your MIB definition files (uncompiled text format) into the **Mibs** directory in the workspace.

   Important You must copy the individual files because subdirectories under the **Mibs** directory are not supported.

   The default paths are:

   Linux:

   ```
   ~/itest/ workspace/resources/SNMP/Mibs
   ```

   Windows:

   ```
   C:\Documents and Settings\<user_name>\My
   Documents\iTest_<version>\resources\SNMP\Mibs
   ```

2. The following steps are optional, but strongly recommended.

   To make it easier to select MIBs from lists during SNMP sessions, you can specify an alias for your proprietary set of MIB variables. For example, when you specify the alias name **ACME** to replace **iso.org.dod.internet.private.enterprises.acme**, the OIDs for the

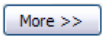
MIB variables in the session window's drop-down lists become much easier to read and select:

```
ACME::IMAGE-MIB
```

appears in the list instead of:

```
iso.org.dod.internet.private.enterprises.acme.IMAGE-MIB
```

Follow these steps:

3. Open the device's session profile and click 

to view the **Session Properties** section.
4. In the tree, click **SNMP MIB Browser > Aliases**.
5. Uncheck the **Inherited values** box (to enable you to specify non-default aliases for proprietary MIBs).
6. Click  to add an alias.
7. In the **Name** text box, specify a name for the alias. In the example, the **Name** is **ACME**.
8. In the **Content** text box, type or paste the OID prefix that you want to alias. In the example, the **Content** is:
9. iso.org.dod.internet.private.enterprises.acme
10. Repeat for each group of MIBs that share an OID prefix
11. Click **Save**.
12. Exit and restart iTest.

SNMP Console

The SNMP Console displays a text log of all traps received by iTest. The same information appears in table format in the SNMP Traps view.

Double-click the tab to maximize the view. Double-click again to minimize it.

SNMP Console Toolbar

Clear Console 	Delete all information from the view.
Scroll Lock	Keep the current text in view, even if new data arrives. Data is added, but scrolling is temporarily disabled.
Pin Console	Place the console view in a different location on the window.
Display selected console	Display the selected instance of the SNMP Console.
Open Console	Open a new instance of the SNMP Console.

Configuring trap settings

See “Configuring trap settings” on page 964.

Example content

```
SNMP trap daemon successfully started (167)
SNMP trap daemon successfully started (163)
SNMP trap daemon successfully started (166)
SNMP trap daemon failed to start on port 162: Address already in use: Cannot bind
SNMP Trap (port=163):
Properties Value
------------------------------------------------------------------------------
--
Source 127.0.0.1
Version Version1
Time Thu Sep 20 20:45:58 NOVST 2007
Community private
Enterprise .1.3.6.1.4.1.1824
OID Type Value
------------------------------------------------------------------------------
---
.1.3.6.1.4.1.1824.1.0.0.1 OctetString "Hello WORLD"
.1.3.6.1.4.1.1824.1.0.0.2 OctetString "Aliens is HERE Trap occurred"
.1.3.6.1.4.1.1824.1.0.0.3 Counter32 3345556
.1.3.6.1.4.1.1824.1.0.0.4 Gauge32 12343212
.1.3.6.1.4.1.1824.1.0.0.5 Integer 99
.1.3.6.1.4.1.1824.1.0.0.6 IpAddress 100.200.123.65
.1.3.6.1.4.1.1824.1.0.0.7 Object Identifier .1.2.3.4.5.6.7.8.9
.1.3.6.1.4.1.1824.1.0.0.8 TimeTicks 6 hours 12 minutes 11 seconds
SNMP Trap (port=166):
Properties Value
------------------------------------------------------------------------------
---
Source 127.0.0.1
Version Version1
Time Thu Sep 20 20:46:22 NOVST 2007
Community private
Enterprise .1.3.6.1.4.1.1824
OID Type Value
------------------------------------------------------------------------------
---
.1.3.6.1.4.1.1824.1.0.0.1 OctetString "Hello WORLD"
.1.3.6.1.4.1.1824.1.0.0.2 OctetString "Aliens is HERE Trap occurred"
.1.3.6.1.4.1.1824.1.0.0.3 Counter32 3345556
.1.3.6.1.4.1.1824.1.0.0.4 Gauge32 12343212
.1.3.6.1.4.1.1824.1.0.0.5 Integer 99
.1.3.6.1.4.1.1824.1.0.0.6 IpAddress 100.200.123.65
.1.3.6.1.4.1.1824.1.0.0.7 ObjectIdentifier .1.2.3.4.5.6.7.8.9
.1.3.6.1.4.1.1824.1.0.0.8 TimeTicks 6 hours 12 minutes 11 seconds
SNMP Trap (port=166):
Properties Value
------------------------------------------------------------------------------
---
Source 127.0.0.1
Version Version2
Time Thu Sep 20 20:47:52 NOVST 2007
Community private
RequestID 1190296072
ErrorStatus 0
ErrorIndex 0
OID Type Value
------------------------------------------------------------------------------
---
.1.3.6.1.4.1.1824.1.0.0.1 OctetString "Hello WORLD"
```

```
.1.3.6.1.4.1.1824.1.0.0.2 OctetString "Aliens is HERE Trap occurred"
.1.3.6.1.4.1.1824.1.0.0.3 Counter32 3345556
.1.3.6.1.4.1.1824.1.0.0.4 Gauge32 12343212
.1.3.6.1.4.1.1824.1.0.0.5 Integer 99
.1.3.6.1.4.1.1824.1.0.0.6 IpAddress 100.200.123.65
.1.3.6.1.4.1.1824.1.0.0.7 ObjectIdentifier .1.2.3.4.5.6.7.8.9
.1.3.6.1.4.1.1824.1.0.0.8 TimeTicks 6 hours 12 minutes 11 seconds
```

SNMP Traps view

The SNMP Traps view displays a table of information on all traps received by iTest from any agent.

The same information appears in text log format in the SNMP Console.

Timestamp	Time at which the trap occurred.
Source	Source (generator) of the trap.
OID	Object ID of the MIB. Example: .**1.3.6.1.4.1.1824.1.0.0.1**
Type	Type of returned data. Example: **IpAddress**
Value	Trap value. Example: **10.155.34.56**

Configuring trap settings

See “Configuring trap settings” on page 964.

Tip On the **Preferences** page, you can configure iTest to open the SNMP Traps view when a trap is received. See “Setting preferences for monitoring SNMP traps” on page 982.

Setting preferences for monitoring SNMP traps

At any time, whether a iTest SNMP session is running or not, iTest receives traps on the ports specified in the iTest preferences. To configure a port to listen for traps during a iTest SNMP session, see “SNMP MIB Browser > Traps” on page 973 and “SNMP MIB Browser > Step Defaults > Traps” on page 978.

To view or edit SNMP preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Session Types > SNMP**. (General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.)

Privileges for SNMP traps

Executing the SNMP daemon on certain operating systems like Linux requires root privileges. Typically, however, you execute iTest as a regular user. As a result, you may not have privileges to listen to the default SNMP trap port 162. In this case, set a different trap port to listen on in either the SNMP session profile or in the iTest SNMP preferences settings. You must configure your SNMP agent to send traps on the new port

- **To configure the ports to monitor for traps**

The **Trap Configuration** section of the page lists the ports that iTest is currently monitoring for traps. iTest listens for the listed traps regardless whether or not a iTest session is running. You

can add, edit, or remove configuration settings. By default, iTest listens on port 162 for non-V3 traps.

To add a port to listen on, click **New** and specify the following settings on the **Configure SNMP Trap** dialog box:

Port	Specify the port to listen on for traps.
V3 Trap	Check the box if the SNMP protocol version of the agent is V3 Default: unchecked
Note The remaining properties apply only for SNMP V3	
Engine ID	Specify the snmpEngineID.
User name	Specify the user name.
Security level	Specify the security level. Default: **No authentication, no privacy**
Authentication algorithm	**Note** This property applies only if you specify a **Security Level** of **Authentication, no privacy** or **Authentication, privacy** Specify the authentication password. Specify the authentication algorithm. Default: **MD5**
Privacy	**Note** This property applies only if you specify a **Security Level** of **Authentication, privacy** Specify the privacy password. Specify the encryption standard. Default: **DES**

- ### To configure general behavior for SNMP traps

The following settings control iTest behavior when traps are received:

Open SNMP Traps console when a trap is received	Open the SNMP Traps console in iTest when a trap is received. Default: unchecked
Open SNMP Traps view when a trap is received	Open the SNMP Traps view in iTest when a trap is received. Default: unchecked
Maximum number of traps on each port	Specify the maximum number of traps (for each port) to list in the SNMP Traps view. When the number of traps reaches the limit, then the oldest trap messages are deleted. Default: 100

CHAPTER 54

Spirent Avalanche sessions

About Avalanche data

- iTest does not use the Spirent Avalanche session steps to analyze the traffic data—other steps in your iTest test case can do that.
- iTest saves all Avalanche responses as structured data. iTest auto-maps responses and auto-generates queries for response data (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create Avalanche response maps and can immediately write analysis rules that use the auto-generated queries.

Options for performing Avalanche tests

Notes Avalanche is supported on Microsoft Windows and on Linux. See "Setting up Avalanche Automation on Linux" on page 1011.

iTest Avalanche sessions rely on a Tcl interpreter to do their work. For more information, see "Tcl Interpreter" on page 1040.

There are several options for running an Avalanche test from iTest:

- **Demo mode**

  In Demo mode, iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. The intent is that you can run a session to learn more about how Avalanche sessions operate on iTest without having to install or run any Avalanche software. See "Running a test using Demo mode" on page 1007.

- **Run the test on a TestCenter device**

  In this option, iTest use the **config.tcl** configuration script generated by Avalanche. Rather than using the Avalanche-generated **test.tcl** script, iTest creates a script based on settings that you specify for properties in the iTest session profile.

  This mode of operation has the advantage that several test settings are parameterized. Parameterization enables you to change settings before starting an automated session and to change them during an interactive session (using the **Configure** button).

  See "Running an Avalanche test on a TestCenter device (Normal mode)" on page 1008

- **Pass-through mode**

  You can execute the Avalanche test using the Tcl test scripts that were previously generated by Avalanche. This setting enables you, for example, to use the latest Spirent cards and appliance models.

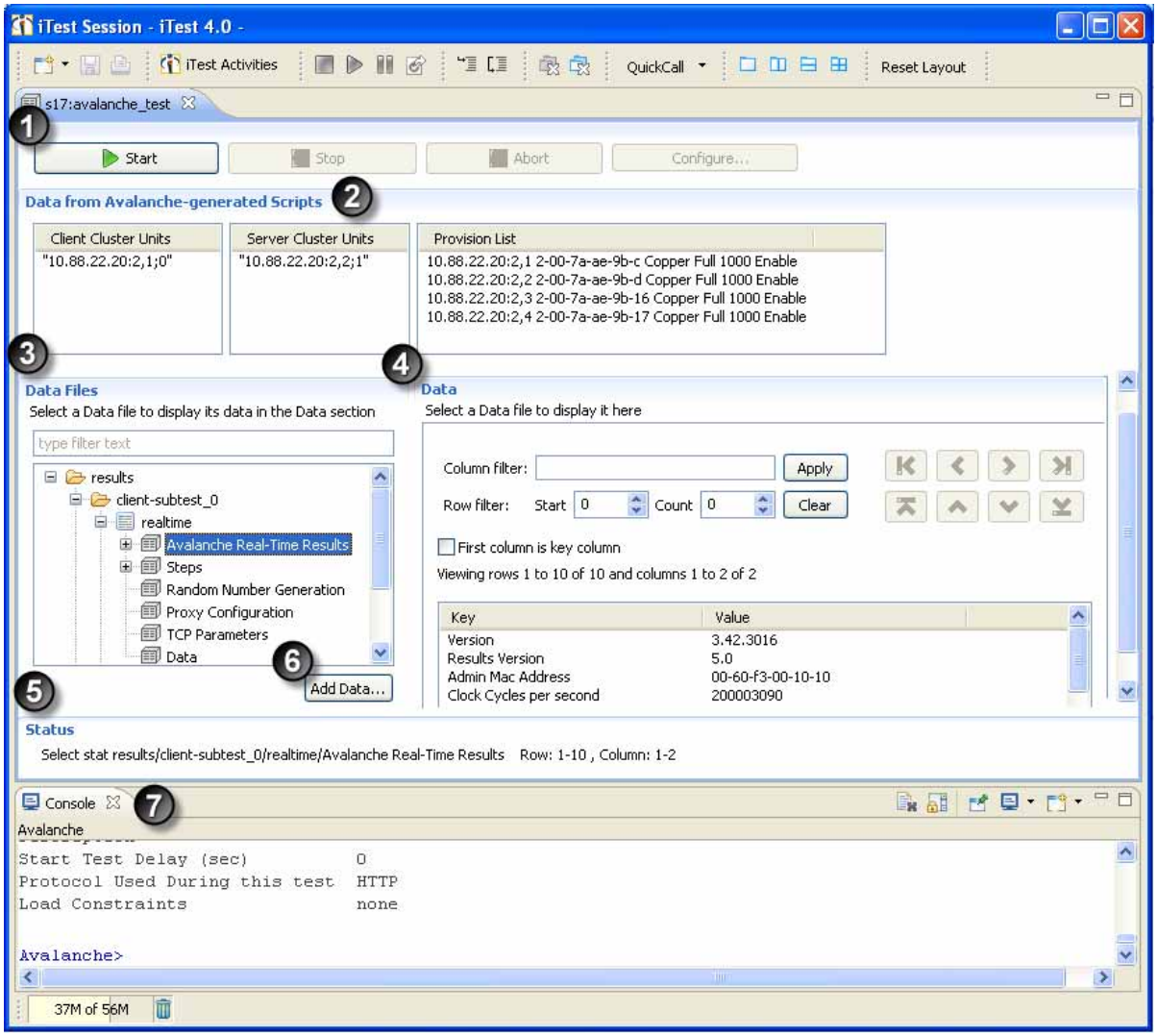
See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009

- **Run the test using Av Tcl scripts**

  See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009

Spirent Avalanche session window

The Avalanche session window is an interactive dashboard where you manage Avalanche operations and traffic and can monitor Avalanche responses. iTest captures all commands and responses and you can save captured items as test case steps that configure, control, and request statistics from Spirent Avalanche devices



1 Test Control section

The buttons in this section control test operation.

Start	Start the Avalanche test. iTest captures a **start** action.
Stop	Stop the Avalanche test. The test ends and returns results. iTest captures a **stop** action. The button is disabled if you specify **Use Avalanche Tcl test files** for the session. See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009 for details.
Abort	The test ends and does not return results. iTest captures an **abort** action.
Configure	The configuration settings for the current session are taken from the configuration script that was specified in the session profile or device. Click **Configure** to change any number of parameter settings for the duration of the current session only — the configuration script is not modified. The button is disabled if you specify **Use Avalanche Tcl test files** for the session. See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009 for details. **Changing a setting** 1. Click **Configure**. 2. In the **Configure Test Parameters** dialog box, select the parameter value in the **Value** cell and type the new value. Change as many values as needed. Parameters are described in "Parameters (Available when you click Configure)" on page 1004. To find a particular parameter quickly, you can type a search string in the filter text box at the top. You can use the * wildcard character. Only parameters with matching text then appear in the list. Click **Clear** to remove the filter text. 3. Click **OK** to apply the settings. iTest captures a single **setParameter** action that sets all the values. To revert to the settings that were in place when you opened the **Configure Test Parameters** dialog box, click **Restore Defaults**. Once you click **OK**, you cannot click **Restore Defaults** to revert to earlier settings.

Parameters (Available when you click* Configure*)

OutputDir	Directory to output the test results to
TrialIfNoLicense - 0 | 1	**0**: Do not use a trial license in the case that no standard license is found **1**: Use the trial license in the case that no standard license is found
TestFile	Filename of archived test
ProjectName	Used when **IsCompact** is **0** Name of the project to be created
TestName	Used when **IsCompact** is **0** Name of the test to be created
ProjectVersion	Used when **IsCompact** is **0** Version of project to be created
TestType	Used when **IsCompact** is **0** Type of test to be created
Ports	List of port addresses for client and server, separated by space characters. **Note** Ports specified in the session properties have higher priority Port addresses use the following format: **ip/slot/port** **{ip/slot/port mode}** (for performance modes)
IsCompact - 0 | 1	**0**: Create and configure a new test **1**: import the test from the spf file **Note** The spf file is generated in Avalanche Commander with **compacted** option
License	Name of current license on the current chassis
Trial - 0 | 1	**0**: Run test in normal mode **1**: Run test in trial mode
Username	Username to use when logging in. Leave blank to use default user name
ReserveForce - 0 | 1	**0**: Do not perform reserve force **1**: Perform reserve force
KeepTest - 0 | 1	**0**: Delete imported test and project after test has finished **1**: Do not delete imported test and project after test has finished
ShowInteractive - 0 | 1	iTest displays interactive event messages in the Console view. You have the option to log the events to a file. **0**: Do not show interactive event messages in log **1**: Show interactive event messages in log

Profiles	Specify an IP address and slot to switch the profile **Tcl syntax** List {<IPAddress> {<slotNumber>, <profileName>; etc}} **Note** Spirent recommends that you do not change the profile using the scripts in the test.tcl file because using the script involves rebooting (which can take several minutes).
SetProfileForce - 0 | 1	**0**: Do not force set profile **1**: Force set profile

2 Information on Cluster Units and Provision List

The tables display the Client and Server clusters from the **test.tcl** script, as well as the provision list from the configuration script (**config.tcl**). The information appears if you configure the session to use the Avalanche-generated Tcl test scripts. This option ignores the iTest session profile settings and executes the Avalanche scripts directly. This enables you, for example, to make use of the latest Spirent cards or appliance models. See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009.

3 Data Files section

The **Data Files** tree displays the results of Avalanche test execution: the folders, files, and data sections from a specified folder.

When you select a file in the tree, iTest displays the data from the file in table format in the **Data** section and as text in the Console view. In addition, if the table is not empty, iTest captures a **selectTable** action.

Tips

- Make the tree wider or narrower as needed by dragging the control between the **Data Files** section and the **Data** section.
- You can run sessions in Demo mode, where iTest displays prepared data, enabling you to view results as if it were being returned by a device. See the descriptions of the **Demo mode** settings in the session profile: "Demo Mode" on page 1038.

4 Data section

When you select an item in the **Data Files** tree, iTest displays the table data in the **Data** section and captures a **selectTable** action. You can filter (limit) the data in various ways to make it easier to view the data and to limit the portion of the data that must be searched by an analysis rule in an automated test case.

Filtering data

By default, one “page” of data (10 columns and 100 rows) appears at a time. You can use the row and column filter features to display a subset of the table data.

If you select a new file in the **Data Files** tree, iTest resets the filter settings and captures a **selectTable** action.

Column filter	Specify filter text to display a subset of table columns. When you click **Apply**, iTest captures a **filter** action with the appropriate **Column filter** property setting. • The filter is case-sensitive. • The **?** wildcard character matches any single character, and ***** matches any number of characters. Because the <space> character is used to separate column titles, you must use a wildcard character to represent a <space> character. **Syntax** To display two columns: <column1Title><space><column9Title> To display all columns between 1 and 9, use **...** <column1Title>**...**<column9Title> **Example** Column title to filter: **Desired Load (SimUsers)** Use the following filter text: **Desired?Load?(SimUsers)** or **Desired*Load*(SimUsers)**
Row filter **Start** **Count**	Use the **Start** and **Count** properties to limit the display to particular rows within the table data. When you click **Apply**, iTest captures a **filter** action with the appropriate **Start row** and **Row count** property settings. **Start**: Specify the number of a particular row at which to start the data display. If no value is specified, then the display starts with the first row of data. **Count**: Specify the number of rows to display. If no value is specified, then all rows are displayed.
vertical paging 	Because Avalanche data sets are very large, iTest displays one “page” of data at a time (10 columns by 100 rows). Use the vertical paging buttons to move one page at a time. iTest captures the appropriate paging action: **pageFirst** **pageUp** **pageDown** **pageLast**
horizontal paging 	Because Avalanche data sets are very large, iTest displays one “page” of data at a time (10 columns by 100 rows). Use the horizontal paging buttons to move one page at a time. iTest captures the appropriate paging action: **pageLeftMost** **pageLeft** **pageRight** **pageRightMost**

Apply	Click **Apply** to apply the specified row and column filter settings to the table. iTest captures a **filter** action with appropriate row and column property settings.
Clear	Click **Clear** to clear the **Column filter** and **Row filter** values and display the entire table. iTest captures a **clearFilter** action.
First column is key column	Check the box to ensure that the first column of the table will always be visible when you move to another page in the table. iTest captures a **lockColumn** action.

5 Status

The **Status** section displays current test status, for example:

```
Opening Avalanche session
Viewing columns 10 to 20 of 55 and rows 115 to 125 of 1270
```

6 Add Data button

Click **Add Data** to specify an existing folder of data files that iTest will add to the **Data Files** list. You can then select any of the files in the folder and work with the data in the normal way.

7 Console

While an Avalanche test is executing, the **Console** view displays device and test status and real-time data as Avalanche returns it. If you enabled logging (with the **Log errors** property in the session profile), then the status text is written to the log file.

After execution finishes, the **Console** view displays the data from any file that you select in the **Data Files** section.

Running a test using Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. The intent is that you can run a session to learn more about how Avalanche sessions operate on iTest without having to install or run any Avalanche software.

Note Demo mode is not supported if you run a test using the Tcl scripts generated by Avalanche.

1 While configuring a session profile in the Session Profile editor, click More >> to view the property tree and select **Demo Mode**. Set the following property values:

Run session in demo mode	Check the box to run the session against prepared data. **Important** If you specify demo mode, then you must specify a value for each of the required properties in the **Session Properties** group, as described in "Session Properties" on page 1034. (Any settings will do — iTest does not use the values. The values are required only to satisfy the iTest validation checks.) Default: unchecked
Results folder	Optional. if **Run session in demo mode** is checked, you can specify the folder from which to obtain the files for display in the **Data Files** list on the session window. If the field is blank, then iTest uses the embedded results from the Avalanche plug-in.

- Save the session profile and start the test.

Running an Avalanche test on a TestCenter device (Normal mode)

In this option, iTest use the **config.tcl** configuration script generated by Avalanche. Rather than using the Avalanche-generated **test.tcl** script, iTest creates a script based on settings that you specify for properties in the iTest session profile.

This mode of operation has the advantage that several test settings are parameterized. Parameterization enables you to change settings before starting an automated session and to change them during an interactive session (using the **Configure** button).

1 You first define a test in Avalanche and then use the Generate Tcl Test utility to save the configuration script (**config.tcl**) and test.tcl script as you normally would. Avalanche uses the script to configure the device for the test.

2 Now configure the iTest session profile:

- Specify the location of the **config.tcl** script that you just created using Avalanche
- Specify property settings that will be used to create a **test.tcl** script for the session

3 Now, start the Avalanche session in iTest, either interactively or as part of an automated test. When the session opens, iTest loads the configuration script to the device (you have the option to change configuration parameter settings before starting the Avalanche test).

4 Upon receiving your **start** command, iTest uses the checks out libraries, connects to the device, and then starts the Avalanche test (checkout the license, configure and reserve ports, generate test files, upload the test to the server and client clusters, and then start test execution). iTest captures the real-time data and displays it in the Console view. The Avalanche test produces responses and data files that you can post-process.

5 When test execution finishes, iTest displays the contents of the **results** directory as a tree in the **Files** section of the Avalanche session window. iTest supports multiple Avalanche sessions within a single instance of iTest (sessions must not use the same cards and ports).

Changing test configuration during an interactive session

Note The operations discussed in this section are not available if the session is executing the Avalanche-generated test scripts.

The configuration settings for the current session are taken from the configuration script (**config.tcl**) that was specified in the session profile. Click **Configure** to change any number of settings for the duration of the current session only — the configuration file is not modified.

Changing a setting

1 Click **Configure**.

2 In the **Configure Test Parameters** dialog box, select the parameter value in the **Value** cell and type the new value. Change as many values as needed.

Tip To find a particular parameter quickly, you can type a search string in the filter text box at the top. You can use the * wildcard character. Only parameters with matching text then appear in the list. Click **Clear** to remove the filter text.

3 Click **OK** to apply the settings. iTest captures a single **setParameter** action that sets all the values.

- To revert to the settings that were in place when you opened the **Configure Test Parameters** dialog box, click **Restore Defaults**. Once you click **OK**, you cannot click **Restore Defaults** to revert to earlier settings.

Executing Avalanche-generated ‘Tcl test’ scripts directly (Pass-Through Mode)

You can execute the Avalanche test using the Tcl test scripts that were previously generated by Avalanche Commander. This setting enables you, for example, to use the latest Spirent cards and appliance models.

Limitations

- Demo mode is not supported.
- The **Abort**, **Stop**, and **Configure** actions are disabled in this mode. Aborting a test can result in stale data in the Tcl test directory.
- Do not close the Avalanche session window while the test is running — wait until the test finishes. It can take several minutes to stop the test, shutdown the ABL server, and close the session.

◆ To execute the Avalanche-generated ‘Tcl test’ scripts

1 Set the following environment variables:

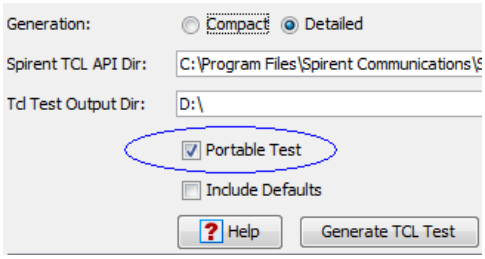
- **SPIRENT_TCLAPI_ROOT**: Path to the directory holding the Tcl API
- **SPIRENT_TCLAPI_LICENSEROOT**: Path to the directory holding the license key file

2 Define a test in Avalanche as you normally would and then use the **Generate Tcl Test** option to save the configuration script (**config.tcl**) and **test.tcl** script in a Tcl test folder.

Pay attention to the following values in **config.tcl**:

OutputDir — The directory that will contain the results. If the directory does not exist, Avalanche creates it in the folder that contains the Tcl files.

IsPortable — In Avalanche Commander, if the **Portable Test** option is checked when you generate the Tcl files, then the value of **IsPortable** is **1**.



TclAPIRoot: — In Avalanche Commander, if the **Portable Test** option is not checked when you generate the Tcl files, then the value of the **TclAPIRoot** variable is used.

If API root directory does not exist or is not specified by the **SPIRENT_TCLAPI_ROOT** environment variable, then iTest will set the value of **TclAPIRoot** to the latest Avalanche API folder. You have the option to set the value as follows:

In the Session Profile editor, click More >> to view the property tree and then select **Tcl**. Specify the path for the **Avalanche Tcl API directory** property.

Note If you move the Avalanche-generated "Tcl test directory" that holds the **test.tcl** and **config.tcl** scripts, then you must:

a Move the entire folder

b Edit **test.tcl**: Specify the new path in the "`set testDirectory`" line.

c Specify the new path in the **Tcl test folder** property in the session profile.

---

3 Configure the session profile. In the session profile editor:

a Check the **Use Avalanche Tcl test files** check box

b For the **Tcl test folder** property, specify the path to the test.tcl and config.tcl files.

4 Save the session profile.

5 Now, open the Avalanche session in iTest, either interactively or as part of an automated test.

When the session starts, iTest checks out libraries, connects to the device, and then starts the Avalanche test (checkout the license, configure and reserve ports, generate test files, upload the test to the server and client clusters, and then start test execution). iTest captures the real-time data and displays it in the Console view. The Avalanche test produces responses and data files that you can analyze when the test concludes.

The session window indicates that the session is executing the Avalanche Tcl test files. On the session window:

- The **Configure** and **Stop** buttons are disabled
- Read-only tables display the Server and Client **clusters** from **test.tcl** and the **Provision List** from **config.tcl**.

When test execution finishes, iTest displays the contents of the **results** directory as a tree in the **Files** section of the Avalanche session window. iTest supports multiple Avalanche sessions within a single instance of iTest (sessions must not use the same cards and ports).

Setting up Avalanche Automation on Linux

Follow the instructions in these topics to set up Avalanche Automation on Linux.

- "Preparing the system" on page 1011
- "Installing Java Manually" on page 1013

Prerequisites

Before you proceed, make sure you have access to the following:

- TCL 8.5. x thread binary (suggested: ActiveState).
- Java Virtual Machine (JVM). OpenJDK or the official JVM from Oracle. Supported versions: 6, 7 for 32-bit (version 8 and 64-bit versions are not currently supported)
- The Avalanche API for Linux (you can ge this from Support website under Avalanche/RHEL (will also work with Debian).

  The example uses a `sudoer` user called **bench**.

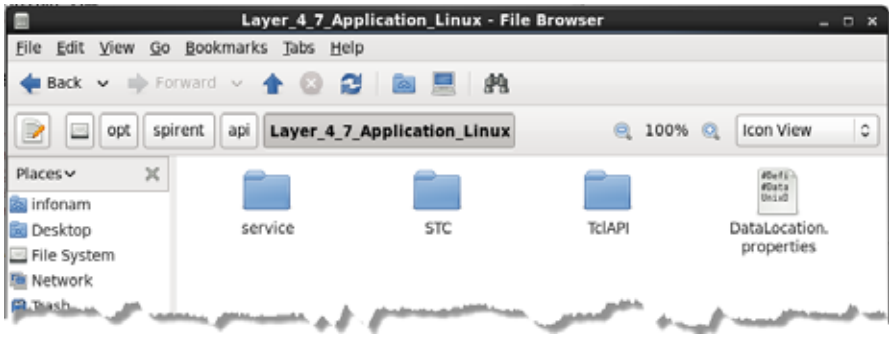
Preparing the system

1 Install the TCL shell and extract the Avalanche API under the **/opt** folder.

```
$ tar -xvzf ActiveTcl8.5.17.0.298612-linux-x86_64-threaded.tar.gz[…..]
$ sudo ./ActiveTcl8.5.17.0.298612-linux-x86_64-threaded/install.sh
[stuff happening, use all default values]
```

2 Extract the API and to **/opt/spirent/api** folder

```
$ tar -xvzf Layer_4_7_Auto_Linux_4.46.tar.gz
$ sudo mkdir /opt/spirent/
$ sudo mkdir /opt/spirent/api
$ sudo mv -f Layer_4_7_Auto_Linux_4.46/Layer_4_7_Application_Linux/
/opt/spirent/api/
```



3 Create these directories.

- Create a directory to store licenses of the load generators.

```
$ sudo mkdir /opt/spirent/licenses
```

Note Skip this step if you use an appliance other than the C1 or C100-MP.

- Create a directory to store all the tests (this can be anywhere of your liking, I'm putting it under the home of the user).

```
$ mkdir ~/spirent/
$ mkdir ~/spirent/tests
```

4 Add the directory path to the TCL interpreter in your **PATH** environment variable:

- Edit file: `$ vi ~/.bash_profile`
- Append the path to Active TCL 8.5 as follows.

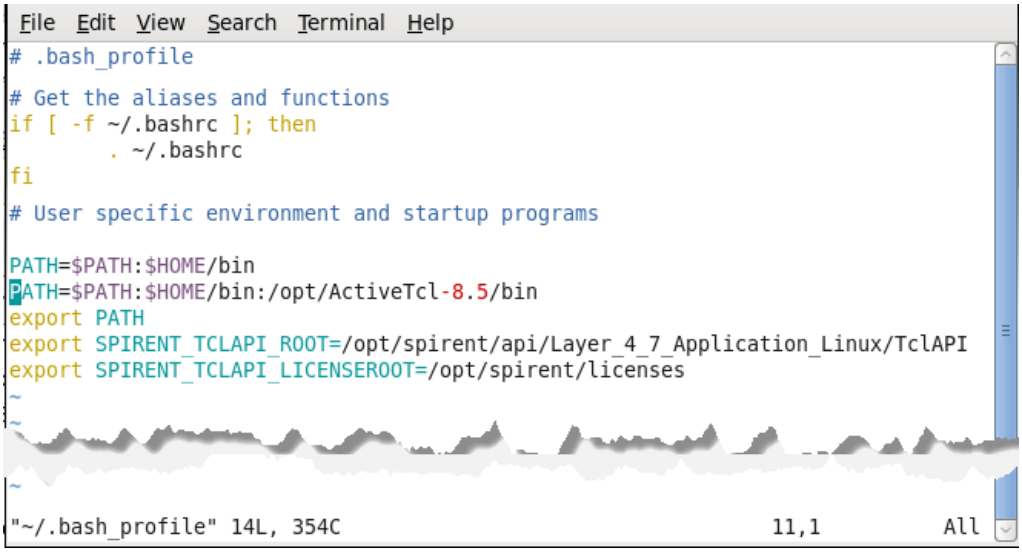
```
PATH=$PATH:$HOME/bin:/opt/ActiveTcl-8.5/bin
export PATH
```

- Reload the profile to make sure it works.

```
$ source ~/.bash_profile
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/bench/bin:/home/bench/bin:/opt/ActiveTcl-8.5/bin
```

5 Edit the user profile file and add the environment variable (that Avalanche TCL scripts rely on) at the end of the file.

```
export SPIRENT_TCLAPI_ROOT=/opt/spirent/api/Layer_4_7_Application_Linux/TclAPI
export SPIRENT_TCLAPI_LICENSEROOT=/opt/spirent/licenses
```



6 Install the Java Runtime Environment (JRE) version (for example, 1.6. OpenJDK).

```
$ sudo yum install java-1.6.0-openjdk.x86_64
```

7 Install the Session Manager daemon. This is a middleware that interfaces the GUI/TCL scripts and the Avalanche backend. Its main function is to ensure backward compatibility of scripts and track the sessions between a front end (TCL, GUI) and the backend.

Create a Shell script as follows to set the required commands.

Note Create an executable script and run this as **root** or a **sudoer**.

```
# cd /opt/spirent/api/Layer_4_7_Application_Linux/service/bin/
# chmod +x ./*
# ./installDaemon.sh
[…]
# ./startDaemon.sh
```

Note An indication that the script was setup correctly is when you see the daemon listening on port 9194:

```
# netstat -tap | grep 9194
tcp 0 0 *:9194 *:*
```

Installing Java Manually

Go to the following location and **download jdk-7u79-linux-i586.rpm**

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html.

Java SE Development Kit 7u79

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	130.4 MB	jdk-7u79-linux-i586.rpm
Linux x86	147.6 MB	jdk-7u79-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u79-linux-x64.rpm
Linux x64	146.4 MB	jdk-7u79-linux-x64.tar.gz
Mac OS X x64	196.89 MB	jdk-7u79-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.79 MB	jdk-7u79-solaris-i586.tar.Z
Solaris x86	96.66 MB	jdk-7u79-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.67 MB	jdk-7u79-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u79-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140 MB	jdk-7u79-solaris-sparc.tar.Z
Solaris SPARC	99.4 MB	jdk-7u79-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u79-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.4 MB	jdk-7u79-solaris-sparcv9.tar.gz
Windows x86	138.31 MB	jdk-7u79-windows-i586.exe
Windows x64	140.06 MB	jdk-7u79-windows-x64.exe

1 Install the RPM file.

```
sudo rpm -ivh filename.rpm
```

where filename is the name of your rpm file (for example, **jdk-7u9-linux-i586.rpm**).

```
Set environment: JAVA_HOME=/usr/java/jdk1.7.0_79
```

2 Append **JAVA_HOME to PATH**

*sudo Alternatives --install /usr/bin/java java /usr/java/*jdk1.7.0_79/bin/java 100

Sudo alternatives --config java

```
File  Edit  View  Search  Terminal  Help
Enter to keep the current selection[+], or type selection number: 6
[root@RHEL6 khoang]# alternatives --install /usr/bin/java java /usr/java/jdk1.7.
0_79/bin/java 100
[root@RHEL6 khoang]# alternatives --config java

There are 7 programs which provide 'java'.

  Selection    Command
-----------------------------------------------
   1           /usr/lib/jvm/jre-1.7.0-openjdk.x86_64/bin/java
   2           /usr/lib/jvm/jre-1.6.0-openjdk.x86_64/bin/java
*  3           /usr/java/latest_jre_x64/bin/java
   4           /usr/java/latest_jre_x86/bin/java
   5           /usr/lib/jvm/jre-1.5.0-gcj/bin/java
 + 6           /usr/java/jdk1.7.0_79/java
   7           /usr/java/jdk1.7.0_79/bin/java

Enter to keep the current selection[+], or type selection number: 7
[root@RHEL6 khoang]# java -version
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) Server VM (build 24.79-b02, mixed mode)
[root@RHEL6 khoang]# tclsh
% info patchlevel
```

3 Select **java jdk 1.7**

4 Restart.

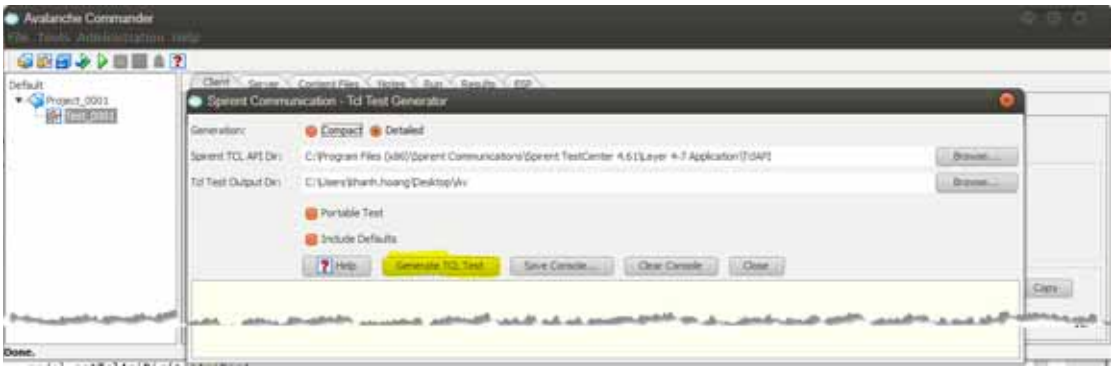
Replay Avalanche test case in Linux

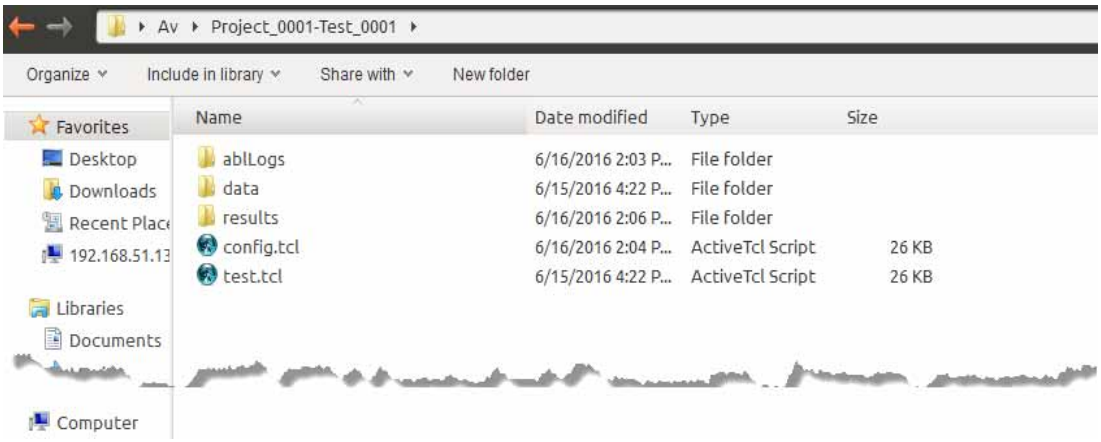
Step 1 Generating TCL Tests

Use any of the following two methods to get Avalanche TCL tests: Write them yourself or export a GUI test to TCL. The example below illustrates exporting a GUI test to TCL.

1 Copy Tcl folder to ***/opt*** folder.

Open an Avalanche Commander session, load a test, and then go to **File > Generate Tcl Test**. The following window displays.



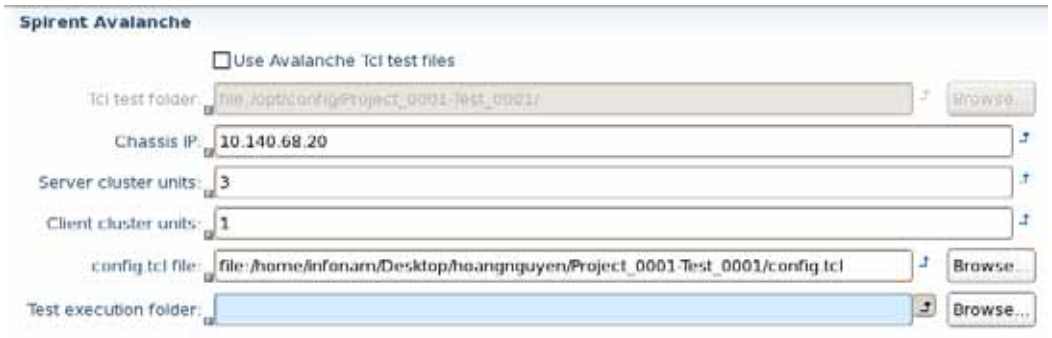


2 Extract **Layer_4_7_Auto_Linux_4.61**

Extract **Layer_4_7_Auto_Linux_4.61** into the folder: **/opt/api/spirent/Layer_4_7_Application_Linux.**

Step 2 Setup Session Avalanche Linux

1 Configure the Chassis IP, Server Cluster / Client cluster unit and especially the config tcl would be the exact config.tcl extracted from a real TCL project.



2 Configure the Tcl tab with the path specified inStep 2 on page 1011.

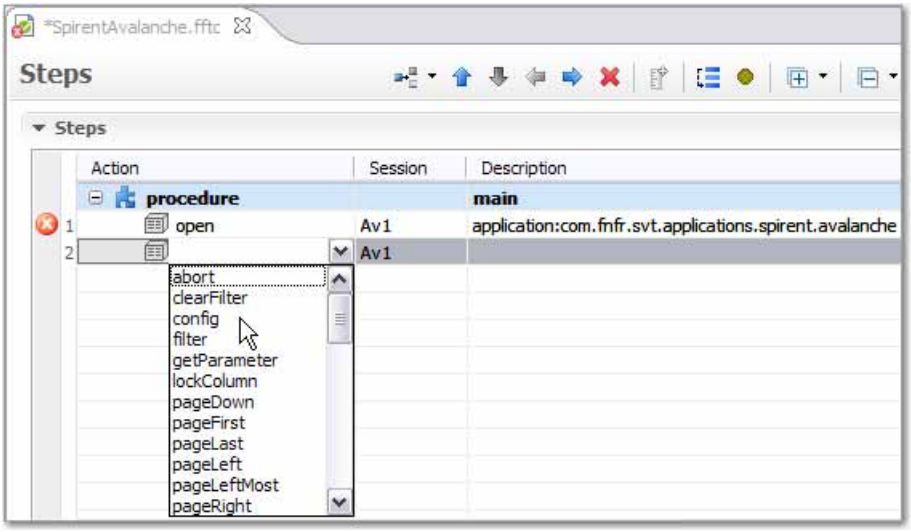
3 As in Windows ("Options for performing Avalanche tests" on page 1001), start the session, replay the test case, and display the report generated with Avalanche in Linux.

Spirent Avalanche command set

This topic describes all Avalanche commands that you can submit from iTest.

1 For a step in an Avalanche session, select one of the listed actions in the **Action** cell.

2 If applicable, specify the arguments in the **Description** cell.



Command reference

Each button on the Avalanche session window correlates to a command.

abort	Ends the test immediately and does not return results.
clearFilter	Clears the **Column filter** and **Row filter** values and displays and returns the entire table of data. See **filter**.
filter	Displays and returns the specified subset of table columns and/or rows. Specify rows and columns in the **Step Properties** section, **Spirent Avalanche filter Properties > Filter Step Properties** page. **Column filter**: Specify filter text to display a subset of table columns. You can use the * wildcard character. Use the **Start row** and **Row count** properties to limit the display to particular rows within the table data. You can use the * wildcard character. **Start row**: Specify the number of a particular row at which to start the data display. If no value is specified, then the display starts with the first row of data. **Row count**: Specify the number of rows to display. If no value is specified, then all rows are displayed. See **clearFilter.**

getParameter	Returns the value of the specified parameter. Specify the parameter name in the **Command** property (**Description** cell). For example, **WrTransactionProfile,Default,Description**
loadResults	
lockColumn	The **lockColumn** action corresponds to the **First column is key** check box on the dashboard, Set **lockColumn** to **true** to ensure that the first column of the table will always be visible when you use one of the **page** actions to move to another page in the table.
pageFirst **pageUp** **pageDown** **pageLast**	Because Avalanche data sets are very large, iTest displays one "page" of data at a time (10 columns by 100 rows). **Interactive dashboard equivalents (vertical paging)** The  buttons in the dashboard perform this group of commands.
pageLeftMost **pageLeft** **pageRight** **pageRightMost**	Because Avalanche data sets are very large, iTest displays one "page" of data at a time (10 columns by 100 rows). **Interactive dashboard equivalents (horizontal paging)** The  buttons in the dashboard perform this group of commands.
selectTable	When you use the dashboard to select a node in the **Data Files** tree, iTest captures a **selectTable** action. Set the **Command** property to the path in the **Results** tree of the table to select. For example: **merged/client/summary/Results Summary**
setParameter	The configuration settings for the current session are taken from the configuration script that was specified in the session profile or device (**config.tcl**). Use the **setParameter** action to change a parameter setting for the duration of the current session only — the configuration file is not modified. Specify the parameter to update and the new value in the **Step Properties** section, **Spirent Avalanche setParameter Properties > setParameter Step Properties** page. **Parameter name:** Type the name of the parameter to change. **Parameter value:** Type the new value for the parameter.
start	Starts to generate traffic on specified ports and starts the Avalanche test.
stop	Stops generating traffic on specified ports and starts the Avalanche test.

Avalanche API Commands

Avalanche Automation provides an Application Programming Interface (API) that you can use to create objects required to run a test.

- A small set of API functions is available to create and manipulate test configurations based on the Avalanche Automation data model. For detailed information about the data model, refer to the *Avalanche Automation Object Reference* document.

- Some helper commands are available to allow the scripter to navigate and utilize Avalanche's data model; for example, to retrieve handle values and verify that a node exists.
- The API defines specific commands that user can use to execute tests.

The Avalanche Automation API commands are:

av_apply

Description

Starts a specified test on the devices.

Syntax

```
av_apply <testHandle> [trial] [continueIfAlreadyRunning] [removeOldTest]
[rerun]
```

Comments

The apply command saves the configuration, performs validation, uploads test configuration to devices, and runs (or reruns) the test. This call is asynchronous, so the client will get control right after the call. The standard async_method_completed event will be sent after the test is started; the specific test state events will also be sent. For more information, please refer to Avalanche™ Automation Programmers' Reference guide.

Return Value

The request id. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_apply $testHandle
```

```
av_apply $testHandle 1 0 1 0
```

Parameters

Name	Type	Description
testHandle	handle	The handle of test to be started.
trial	Boolean (1|0)	If true, then the trial test will be started. If false, then a license validation error may occur.
continueIfAlready Running	Boolean (1|0)	If true, then the test will be started, even there is another test running on the device. If it is the same test, then the system will reconnect to it. Otherwise, the test on the device will be aborted.

removeOldTest	Boolean (1|0)	If true, then the previous test will be removed from the device, before this test starts.
rerun	Boolean (1|0)	If true, then the test will be rerun. This saves time for tests that you have already run (full or trial), because they do not need to be reloaded. If false (default), the test will be run in the normal manner.

av_config

Description

Updates the attribute of an object with a new value, if it meets the validation rules.

Syntax

```
av_config objectHandle -attrName value [[-attrName value] ...]
av_config objectHandle -DANPath value [[-DANPath value] ...]
av_config DDNPath -attrName value [[-attrName value] ...]
av_config DDNPath -DANPath value [[-DANPath value] ...]
```

Comments

The av_config command modifies the value of one or more object attributes, if it meets the validation rules. Note: If you attempt to modify an attribute for a read-only object, or a specified value does not meet the validation rules, the av_config command raises an exception.

- When you modify object attributes, use attrName/value pairs. For example:

  ```
  av_config project1 -name Project1
  ```

- You can use Direct Descendant Notation (DDN) to identify the object and Descendant Attribute Notation (DAN) to identify the attribute. For example:

  ```
  av_config $project.test -name Test1
  ```

  ```
  av_config $project -userprofile.name SSLv3
  ```

A DAN path is a dotted path name beginning with a sequence of one or more object types, and ending with an attribute name. Avalanche Automation combines the handle (or the DDNPath) with the DANPath to resolve the attribute reference. The path must identify a valid sequence of objects in the Avalanche Automation data model hierarchy.

In both DDN and DAN paths, an object type name may have an index suffix (an integer in parentheses) to reference one of multiple children of the same type.

For more information about these notations, see section “Referencing Objects: Object Paths” *Avalanche Automation Programmers’ Reference* guide.

Return Value

None. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_config userprofile1 -dnsRetries 10
```

```
av_config userprofile1 -cifsng.cifsngDataRandomization true
```

```
av_config project.test.userprofile -sipng.firstRTPPort 1026
```

Parameters

Name	Type	Description
objectHandle	handle	Data model object/node handle

attrName value	name/value pair	An attribute name/value pair. The attr portion of the pair is the name of the attribute to be modified. The value portion specifies the new value. You can specify one or more attrName/value pairs in a single function call. The attribute name and value must be separated by a space; each attrName/value pair in a sequence must also be separated by a space.
DANPath	string	A dotted path name that begins with a sequence of one or more object types, and ending with an attribute name. Avalanche Automation combines the objectHandle (or the directDescendantPath) with the descendantAttributePath to resolve the attribute reference.
DDNPath	string	A dotted path name sequence that begins with an object handle, followed by one or more object type names. The path must identify a valid sequence of objects in the data model hierarchy. Avalanche Automation returns data for the object identified by the last name in the sequence. Use index values to identify one of a set of children of the same type. Index values are assigned in the order of creation. An unqualified type name (a name with no index value) indicates the first child object of that type for the parent.

av_connect

Description

Connects to the specified device.

Syntax

```
av_connect ipAddress [-type STC|Appliance] [–executesynchronous
false|true]]
```

Comments

After this command call is completed, the device will be added to the device list under the PhysicalChassisManager object. If Avalanche Automation is already connected to this device, then it gets the latest state from the device and returns the existent handle. The av_connect command runs in synchronous mode by default. The mode can be changed by using the -executesyschronous Boolean option.

If a -type is not specified, the Appliance platform will be assumed and tried. If Avalanche TclAPI fails to connect, then the Spirent TestCenter chassis is assumed to be the hardware platform, and the connection will be retried.

Return Value

The request id, if run in asynchronous mode; the chassis/appliance handle, if run in synchronous mode (default). Otherwise, errors will be displayed on the screen.

Example

```
%av_connect 10.72.55.80
physicalchassis1
When run in synchronous mode, the return value is the handle of the device.

%av_connect 10.72.55.80 –executesynchronous false
279When run in asynchronous mode, the return value is the request id of the
command.

%av_connect 10.34.76.52 –type stc
physicalchassis2
```

Parameters

Name	Type	Description
ipAddress	string	IP address of the Spirent TestCenter chassis or Appliance.

type	string	A string (stc or appliance) that indicates which connection method Avalanche Automation should use.
executesyschronous	Boolean (true|false)	Boolean value that specifies whether to run the command in synchronous or asynchronous mode.

av_create

Description

Creates a new object of the specified type, under the specified parent.

Syntax

```
av_create handle -relationName | -objectTypeName value [[-attr value] ...]
av_create DDNPath -relationName | -objectTypeName value [[-attr value] ...]
av_create handle -relationName | -objectTypeName value [[-DANpath value]
...]
av_create DDNPath -relationName | -objectTypeName value [[-DANpath value]
...]
```

Comments

The av_create command creates one or more Avalanche Automation objects under the specified parent object. When you call the create function, you specify the type(s) of one or more objects to be created. You can specify:

- An object type name (such as the Project object or the Test object). For example:
- av_create project –under system1 -name Project1
- When you create an object, you must specify the handle of the parent object under which the new object is to be created.
- When you create an object, you can also set the object attributes at the same time. To set attributes, specify one or more attribute name/value pairs.
- If you specify attribute name/value pairs, together with an object type path, Avalanche Automation applies the attribute values to the object associated with the last name specified in the object type path. In the following example, Avalanche Automation creates a Project object. When Avalanche Automation creates the Project object, it sets the name attribute to Project1 and the path attribute to C:\Project\Project1.

  ```
  av_create project –under system1 -name Project1 -path
  “C:\Projects\Project1”
  ```

- You can specify a Descendant Attribute Notation (DAN) path as part of the attribute reference. Avalanche Automation uses the specified object type to create the primary object, and the DAN path to create any additional objects. For information about path name specification, see section “Object, Attribute, and Relation References” *in Avalanche™ Automation Programmers’ Reference* guide.

Return Value

The av_create command returns a unique string value that is the object handle for the object specified in the function call. (The av_create function returns only the handle for the primary object that is created. To obtain the handles for any descendent objects that are created, use the get function to retrieve the child objects.)

Example

```
set hProject [av_create project -under system1 -name Project1]

set hTest [av_create tests -under $hProject -name Test1 -testType
deviceComplex ]
```

```
set hServerProfile [av_create ServerProfiles -under $hProject -name
ServerProfile -applicationProtocol HTTP -http.keepAlive on]
```

Parameters

Name	Type	Description
handle	handle	Specifies the handle of the parent for the newly created object.
relationName or objectTypeName	string	The name of the relation from the parent to the created object, or the name of the object's type.
DDNPath		A dotted path name sequence that begins with an object handle, followed by one or more object type names. The path must identify a valid sequence of objects in the data model hierarchy. Avalanche Automation returns data for the object identified by the last name in the sequence. Use index values to identify one of a set of children of the same type. Index values are assigned in the order of creation. An unqualified type name (a name with no index value) indicates the first child object of that type for the parent.
DANpath		A dotted path name beginning with a sequence of one or more object types, and ending with an attribute name. Avalanche Automation combines the objectHandle (or the directDescendantPath) with the descendantAttributePath to resolve the attribute reference.
attr/value		The attr portion of the pair is the name of the attribute to be modified. The value portion specifies the new value. You can specify one or more attr/value pairs in a single function call. The attribute name and value must be separated by a space; each name-value pair in a sequence must be separated by a space.

av_createProject

Description

Creates a new project.

Syntax

```
av_createProject -project name
```

Comments

Creates a new Project object in the system, with a specified name.

Return Value

Handle of the newly created object.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_createProject -project Project1
```

Parameters

Name	Type	Description
name	string	Name for the created Project object

av_createTest

Description

Creates a new Test object under a specified Project object

Syntax

```
av_createTest -project projectHandle -test testName -type testType
```

```
av_createTest -project projectName -test testName -type testType
```

Comments

Creates a new test, under the specified project, with the specified name of a specified type.

Return Value

Handle of the newly created Test object.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_createTest -project project1 -test Test1 -type deviceComplex
av_createTest -project Project1 -test Test2 -type deviceSimple
```

Parameters

Name	Type	Description
projectHandle or projectName	handle string	Handle of the project under which the new Test object is created. Name of the project under which the new Test object is created
testName	string	Name of the test to create.
testType	string	One of the available types of Test objects.

av_delete

Description

Deletes a node from the data model.

Syntax

```
av_delete handle
av_delete DDNPath
```

Comments

Deletes the object identified by the objectHandle or DDNPath from the data model. Avalanche Automation also deletes all descendants of the specified object (if any).

Return Value

None. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_delete $projectHandle
av_delete $projectHandle.userprofiles(1)
```

Parameters

Name	Type	Description
handle	string	The object handle that identifies the object to be deleted.
DDNPath	string	A dotted path name sequence that begins with an object handle, followed by one or more object relation names. The path must identify a valid sequence of objects in your data model hierarchy. Avalanche Automation deletes the object identified by the last name in the sequence. Use index values to identify one of a set of children of the same type. Index values are assigned in the order of creation. An unqualified type name (a name with no index value) indicates the first child object of that type for the parent.

av_disconnect

Description

Disconnects from the specified device and removes it from the device list under the PhysicalDevicesManager object.

Syntax

```
av_disconnect ipAddress
```

Comments

Does nothing, if Avalanche Automation was not connected to specified device.

Return Value

None. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_disconnect 10.50.20.77
```

Parameters

Name	Type	Description
ipAddress	string	The IP address of the Appliance or the Spirent TestCenter chassis.

av_get

Description

Returns the value(s) of one or more object attributes, or a set of object handles.

Syntax

```
av_get handle [-attributeName]

av_get handle [-DANPath]

av_get DDNPath [-attributeName]

av_get DDNPath [-DANPath]

av_get handle | DDNPath [-relationName]
```

Comments

The av_get command returns the value of one or more object attributes, or, in the case of relation references, one or more object handles.

- The handle identifies the object from which data will be retrieved. If you do not specify any attributes, Avalanche Automation returns the values for all attributes and all relations defined for the object.
- The **attributeName** identifies an attribute for the specified object.
- The DANPath (Descendant Attribute Notation path) is a dotted path name beginning with a sequence of one or more relation names, and ending with an attribute name. A relation name may have an index suffix (an integer in parenthesis) to reference one of multiple children of the same type. Avalanche Automation combines the handle (or the DDNPath) with the DANPath to resolve the attribute reference. The path must identify a valid sequence of objects in the test hierarchy. For example:

  ```
  av_get $project test(1).name
  ```

- Avalanche Automation combines the object and attribute specifications to retrieve the value of the attribute for the first Test object child of the $project.
- The DDNPath (Direct Descendant Notation path) is a dotted path name sequence. The sequence begins with an object handle, followed by one or more relation names. The path must identify a valid sequence of objects in the data model hierarchy. Avalanche

Automation returns data for the object identified by the last name in the sequence. For example:

```
av_get $project1.test -name
```

- In this case, Avalanche Automation returns the value of the name attribute for the first Test child of the specified Project object.
- If there is more than one instance of a particular object type, as children of the specified object, use an index notation. (In the example above, the index value 1 is implied.) Avalanche Automation assigns index values in the order of object creation. For example:

```
av_get $project.test(2)
```

- Avalanche Automation returns the attributes and all relations for the second Test object child of the specified Project object.
- When you use a relation reference with the get function, it provides access to one or more objects connected to the object identified by a handle (or DDNPath). Specify a name for the relation reference, using relationName. For example:

```
av_get $hProject -Tests
```

- This function call returns the handle(s) for the Test child object(s) of the Project object.

Return Value

When you retrieve one or more attributes, av_get returns a string containing a single attribute value or a set of name-value pairs. If you do not specify any attributes, the get function can return either a single attribute value or a list of name-value pairs.

When the get function returns a list of name-value pairs, Avalanche Automation returns a single string containing the list. Each attribute name and its value is separated by a space, and the name-value pairs are also separated by a space:

```
attr1 value1 attr2 value2 ..attrN valueN
```

When you specify a relation name, the get function returns one or more handles.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_get $project -path

av_get $test -netrworkprofile.tcpoptions.tcptimeout

av_get $project.userprofile(2) -nfs.dataRandomization
```

Parameters

Name	Type	Description
handle	handle	Identifies the object from which data will be retrieved.
attributeName	string	Identifies an attribute for the specified object.
DDNPath		A dotted path name sequence that begins with an object handle, followed by one or more relation names. The path must identify a valid sequence of relations in your data model hierarchy. Avalanche Automation returns data for the object identified by the last name in the sequence. Use index values to identify one of a set of children of the same type. Index values are assigned in the order of creation. An unqualified type name (a name with no index value) indicates the first child object of that type for the parent.

DANPath		A dotted path name beginning with a sequence of one or more relation names, and ending with an attribute name. Avalanche Automation combines the handle (or the DDNPath) with the DANPath to resolve the attribute reference.
relationName		Specifies name of the relation that should be retrieved from the target object.

av_getEvents

Description

Gets the events that were generated by Avalanche Automation for this session, from the latest av_getEvents call.

Syntax

```
av_getEvents
```

Comments

None.

Return Value

A list of generated events. Refer to section “Events Handling” in *Avalanche Automation Programmers’ Reference* guide for event object details.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_getEvents
```

Parameters

None.

getOrCreateNode

Description

Gets handle of node under specified parent with specified name and relation from parent. If the node does not exist, this function creates it.

Syntax

```
getOrCreateNode parentHandle relationName args
```

Return Value

The getOrCreateNode command returns a handle which is under parent with specified name and relation from parent.

Example

```
set httpbody_handle [getOrCreateNode $projectHandle httpbodies Default]
set tests_handle [getOrCreateNode $testHandle configuration Test_0001]
```

Parameters

Name	Type	Description
parentHandle	handle	Data model object/node handle.
relationName	name	A relation name under parentHandle. Refer to the relation tree in the *Avalanche Automation Object Reference.*
args	string or strings	One string or several strings that define the names of nodes to be created.

av_getSessions

Description

Retrieves the list of registered sessions on Avalanche Automation.

Syntax

```
av_getSessions
```

Comments

You can use this function without being logged in to an Avalanche Automation session.

Return Value

A list of registered sessions.

Example

```
av_getSessions
```

Parameters

None.

av_handleOf

Description

Retrieves the handle of an object by its type name, parent handle, and name of the object.

Syntax

```
av_handleOf parentHandle relationName objectName
```

Comments

Searches children that are identified by the relation name of the parent object, identified by a handle, and retrieves the handle of the object named objectName. The object that is searched should have a name attribute.

Return Value

Handle of the object.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_handleOf project1 serverprofiles IPv6

av_handleOf system1 projects Project1
```

Parameters

Name	Type	Description
parentHandle	integer	Handle of the parent object.
relationName	string	Name of the relation – the search will be performed within the objects related from the parent object by this relation.
objectName	string	Name of the object for which to search (i.e. value of name attribute)

av_login

Description

Starts a new Avalanche Automation session or connects to an already existent session.

Uses the default workspace, named *Default*, if –workspace workspaceName option - value pair has not been defined. Use the -workspace workspaceName option - value pair to log in to an existing custom workspace, or to create a new custom workspace, and log in to it.

Syntax

```
av_login [userName] [password] [mode] [-workspace
workspaceName|-temp-workspace]
```

Comments

The av_login command should be called first to start a session, before retrieving or manipulating the data model with any other Avalanche TclAPI command (the exception is the av_getSessions command).

The <username> argument defines the name of the user that is used by the av_reserve command to reserve ports. If the username is not specified, then the system username defined by the resident Operating System is used.

The av_login command starts a new session, if the specified mode is *manage*, which is also the default mode. Therefore, when no mode parameter is specified, a new session is started. The av_login command connects to an existing session, if the specified mode is *monitor*. (Note: The password parameter is currently ignored.)

To go to monitor mode, you must use the same username/password/workspace parameters specified when you first created the session in manage mode.

Example

To go to manage session:

```
#create session
av_login

#get current workspace name
av_get system1 -workspace
Default
```

To go to monitor session:

```
#create monitor session
#login parameters must be the same as those used to create
#the manage session (user name, password workspace name)
av_login <user_name> <password> monitor -workspace <workspace_name>

# You can confirm that it is the monitor session by using
# av_config or av_create command
av_create project -under system1 -name project_0002
(insufficient_rights) You must manage this session to perform this action
```

Return Value

New session id. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_login
av_login john
av_login john johns_password monitor
av_login –workspace johns_workspace1
av_login john johns_password monitor –workspace johns_workspace1
av_login –temp-workspace
```

Note The av_login command uses workspace *Default* by default, when nothing is defined. If *Default* workspace is already in use by another TclAPI session, or by the GUI (remember that *Default* workspace name is also used by the GUI), then the av_login command will fail with the following error message:

```
ERROR: Cannot use this workspace:
Cannot lock file C:/ProgramData/Spirent/Avalanche <version>/running/<user_name>/Default
```

Parameters

Name	Type	Description
userName	string	Name of the user for which data will be processed. By default, the name of the current user will be used.
password	string	User's password. Currently ignored.
mode	string	Mode of the session. Permitted values are manage or monitor. The default mode is manage.
-workspace	key	Use to log in to an existing custom workspace or to create a custom workspace and log in to it.
workspaceName	string	Workspace name, should be a valid directory name. Mandatory if -workspace key was used.
-temp-workspace	key	Use it to log in to temporary workspace. Warning: The temporary workspace is destroyed with all of your created tests after you log out. Be sure to copy the data you want to save before you log out.

av_logout

Description

Closes the session, stops any running test, saves any non-saved data on the disk, and stops the Avalanche Automation (java) process by default, when either no argument is provided or when the shutdown argument is provided (see examples). The only difference between the shutdown and no-shutdown arguments is stopping or not stopping the Avalanche Automation (java) process. The av_logout command with the no-shutdown argument will leave the Avalanche Automation (java) process running after the user logs out.

If the temporary workspace is used (see av_login command description), the av_logout command deletes all the tests and test results that were created during the session.

Syntax

```
av_logout [shutdown | no-shutdown]
```

Comments

None.

Return Value

None.

Example

```
av_logout
av_logout shutdown
av_logout no-shutdown
```

Parameters

Name	Type	Description
shutdown	string	Default behavior. Avalanche Automation (java) process is stopped.
no-shutdown	string	Avalanche Automation (java) process remains up and running while the user is logged out.

av_nodeExists

Description

Checks whether the object specified by the handle exists in the data model.

Syntax

```
av_nodeExists handle
```

Comments

Checks whether the object specified by the handle exists in the data model (i.e. the get function will return information about the object).

Return Value

1, if the object exists in the data model, otherwise 0.

Example

```
av_nodeExists project1
av_nodeExists $testHandle
```

Parameters

Name	Type	Description
handle	handle	Handle to check

av_normalizePath

Description

Returns concatenation of test.tcl script directory and specified 'path'. av::spiGlobals(ScriptDirectory) is set in test.tcl

Syntax

```
av_normalizePath path
```

Return Value

Concatenation of test.tcl script directory and specified 'path'. av::spiGlobals(ScriptDirectory) is set in test.tcl

Example

```
set action_list_path [av_normalizePath {data/urls/Default}
```

Parameters

Name	Type	Description
path	string	Specified 'path' to be concatenated.

av_perform

Description

Executes a custom command or subcommand for the specified object.

Syntax

```
av_perform sub-command handle [[argument] [...]]
```

Comments

The av_perform command executes a sub-command. See the *Avalanche Automation Object Reference* document for a complete list of sub-commands.

Return Value

The return value depends on the sub-command type, and is absent in most cases.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_perform save system1
av_perform export system1 –projectsTestsHandles test1
```

Parameters

Name	Type	Description
sub-command	string	The sub-command name.
handle	handle	The handle of the object to which the action will be applied.

av_release

Description

Releases the specified port.

Syntax

```
av_release portAddress
```

Comments

You can only release ports that you have reserved. For information about port reservations, and the syntax for identifying ports, see the description of the “reserve function” in *Avalanche™ Automation Programmers’ Reference* guide.

Return Value

None. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_release 10.50.70.82/1/1
```

Parameters

Name	Type	Description
portAddress	string	Port address is specified in form “ip-address/slot/port”, for example “10.20.30.40/1/2”. Both slot and port numbers are 1-based.

av_reserve

Description

Reserves the specified port.

Syntax

```
av_reserve portAddress
```

Comments

Reserves the specified port for the username that was specified/determined upon the av_login command. You must be connected to the chassis (or appliance) before you can reserve ports. The port can only be reserved if it is available (that is, if not disabled or reserved by another user). To *force reserve* a port, use the av_perform reservePort command.

Return Value

Handle to the PhysicalPort object.

Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_reserve 10.50.70.82/2/1
```

Parameters

Name	Type	Description
portAddress	string	Port address is specified in the form ip-address/slot/port, for example: 10.20.30.40/1/2. Both slot and port numbers are 1-based.

av_subscribe

Description

Subscribes to view the list of runtime statistics that users specify.

Syntax

```
av_subscribe side viewAttributesList
```

Comments

Subscribes to view runtime statistics for those that user specifies as viewAttributesList. The attribute names should be one of the supported statistics names, for example, http,successfulConns or http,attemptedConns. Wildcards are also supported, such as http*. Please refer to Avalanche™ Automation Programmers' Reference for full list of runtime statistics.

Return Value

Returns the handle to the ResultDataSet object, which consists of the ResultDataObject with statistics values. By default, the returned ResultDataSet will only contain the latest actual values. In order to obtain the values from a specific point in time during the test run, user must add the 'all' keyword to the list of viewAttributesList. See *Avalanche Automation Programmers' Reference* guide for more information.

Example

```
av_subscribe client http,successfulConnshttp,attemptedConns

av_subscribe server http*
```

Parameters

Name	Type	Description
side	string	Runtime statistics side - server or client.
viewAttriutesList	string	List of the runtime statistics names. It may also include the 'all' keyword, if you want to see the runtime values history for the test.

av_unsubscribe

Description

Removes a subscription for the specified ResultDataSet.

Syntax

```
av_unsubscribe handle
```

Comments

The av_unsubscribe command removes a subscription for the specified handle of the ResultDataSet object that was returned by the subscribe function.

Return Value

None. Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_unsubscribe $rdsHandle
```

Parameters

Name	Type	Description
handle	handle	The handle for the ResultDataSet object associated with the subscription. (The handle is returned by the subscribe function.

av_waitUntilCommandIsDone

Description

Waits until the command specified by the request id is complete.

Syntax

```
av_waitUntilCommandIsDone [requestId]
```

Comments

This function waits until the command, specified by request id, is out of the PENDING state and completes its job. This is useful for asynchronous commands. If the requestId is not specified, then it waits until any command is completed.

Return Value

Result of an asynchronous command.
Errors are raised as exceptions, encoded as string values that describe the error condition.

Example

```
av_waitUntilCommandIsDone [av_connect 10.50.70.82 –executesynchronous
false]
```

CAUTION “av_waitUntilCommandIsDone” will time out if the command it waits for does not complete in a certain time.

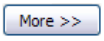
Parameters

Name	Type	Description
requestId	integer	Request id returned by an asynchronous command.

Session profile property settings for Spirent Avalanche sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Using Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. To use Demo mode, set the **Demo mode** properties and provide values for the required properties in the Session Properties group. See "Demo Mode" on page 1038 for details.

Session Properties

The first two properties are associated with the option to execute the test defined in the Avalanche-generated **test.tcl** and **config.tcl** files. See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009.

Use Avalanche Tcl test files	Check the box to execute the test defined in the Avalanche-generated **test.tcl** and **config.tcl** files. See "Executing Avalanche-generated 'Tcl test' scripts directly (Pass-Through Mode)" on page 1009. If you check the box, then you must specify the path to the folder that holds the **config.tcl** and **test.tcl** files in the **Tcl test folder** property Uncheck the box to cause iTest to execute the **test.tcl** script and execute a default **config.tcl** script. When iTest processes **test.tcl**, several settings are parameterized, enabling you to configure them during the session using the **Configure** button. See "Test Control section" on page 1003 for details. Default: unchecked
Tcl test folder	Used only if **Use Avalanche Tcl test files** is checked. Specify the path to the **Tcl test folder** that holds the Avalanche-generated **test.tcl** and **config.tcl** scripts. **Note** This is also the working folder for test execution. All files that are generated during execution (for example, results) are stored in this folder. Default: [blank]

The following property settings are used only if **Use Avalanche Tcl test files** is unchecked. See "Running an Avalanche test on a TestCenter device (Normal mode)" on page 1008.

Chassis IP	Required. Specify the IP address or hostname of the Spirent device.
Client cluster units	Required. Space-separated list of unit IDs for the client. **Note** Be sure to review "Specifying cards, slots, port groups, and ports/virtual ports" on page 1040 before setting values. Each Unit ID has either of the following formats: **<portGroup>** When Avalanche is being run on an appliance, specify the port group number. or **<slotNumber>,<portGroupNumber>;<unitNumber>** **<slotNumber>,<portGroupNumber>** **unitNumber** is the number of the unit on which the port belongs. For Avalanche 4.10 and newer, specify **0** to auto-select the unit number. For configing multiple chassis, enter as follows: Appliance: Client cluster units: chassis/card/port or chassis/port or STC chassis : Client cluster units : chassis/card/group;virtualport The virtualport must match the Virtual port set in the "Port Provision List" on page 1036.
Server cluster units	Required. Space-separated list of unit IDs for the server. See the description for **Client cluster units**
config.tcl file	Required. Specify the path to the TCL initialization script (**config.tcl**) for sessions that use this session profile. The script includes all required parameters for test execution. When iTest processes the script, several settings are parameterized, enabling you to configure them during test execution using the **Configure** button. See "Test Control section" on page 1003 for details. See the **Enable pass-through mode** property.
Test execution folder	Optional.This is the folder whose contents are displayed in the **Data Files** tree in the session window. Specify the URI pointing to the working directory for test execution. All files that are generated during execution (for example, results) are stored in the specified directory. If you do not specify a folder, then iTest creates a temporary folder for this purpose.

Port Provision List

The **Port Provision List** properties are used only for STC mode (device is not an appliance), as described in "Advanced > License" on page 1038.

Tip Be sure to review "Specifying cards, slots, port groups, and ports/virtual ports" on page 1040 before setting values.

To add a port definition:

1. Check **Include additional values from list** to allow you to add a port definition. (For a discussion on inheriting settings from reference session profiles—the **Include inherited values** checkbox—see "Property values: Inheriting settings" on page 62.)
2. Now, click  to add a new port definition. Each new port definition appears in the list.

Specify the following properties.

Card	Required. Specify the card as an integer number Default: 1
Port	Required. Specify the port as an integer number Default: 1
Virtual port	Optional. Specify the virtual port. When specified, the Virtual card of the Server and Client must match with two different virtual port in the **Port Provision list**. You must create 2 ports with different virtual port. Default: 1
MAC address	Required. MAC address that is assigned to the port by the configuration script (**config.tcl**). Use **x-xx-xx-xx-xx-x** format.
Port speed	Required. Specify the port speed in Mbps. Default: 1000
Duplex mode	Required. Specify either **Full-duplex** or **Half-duplex** Default: Full-duplex
Port media	Required. Specify either **Copper** or **Fiber** media. Default: Copper
Auto negotiation	Optional. Check the box to specify auto-negotiation.
Operational mode	Optional. Specify either **func** (Functional) or **perf** (Performance) mode. **Note** Use the **Appliance Operational Mode** property when Avalanche is running on an appliance The 10Gbps AP card has two 10Gbps ports. **Functional mode**: Both ports can be reserved and used. Each port will be assigned with three virtual ports (1,2,3 on each port, for example, : 192.168.1.1:2,1,1 through 192.168.1.1:2,1,3 and 192.168.1.1:2,2,1 through 192.168.1.1:2,2,3). This mode allows a tester to use both ports at maximum of 5Gbps port bandwidth per port. **Performance mode**: Only the first port is activated (for example: 192.168.1.1:2,1) with seven virtual ports. (192.168.1.1:2,1,1 through 192.168.1.1:2,1,7). With only the first port activated, it will achieve 10Gbps bandwidth on the port. Default: func

Appliance Operational Mode

Operational mode	Specify this setting to use when Avalanche is running on an appliance. (Unit numbers for the port group are auto-determined during test execution.) See the Spirent TCP API documentation for information on how the setting relates to CPU and port group usage.

Filter step defaults

Column filter	Optional. Use the **Column filter** property to limit the display to the specified subset of table columns. Specify filter text to display a subset of table columns. You can use the * wildcard character.
Start row	Optional. Use the **Start row** and **Row count** properties to limit the display to particular rows within the table data. Specify the number of a particular row at which to start the data display. You can use the * wildcard character. If no value is specified, then the display starts with the first row of data.
Row count	Optional. Specify the number of rows to display. You can use the * wildcard character. If no value is specified, then all rows are displayed.

Advanced > Debug and Logging

Log errors	Used by the Avalanche API calls. If you check the box, then the execution folder will contain a log file. **Note** Use this property only to resolve problems. If you encounter an unknown error during test execution, you can enable error logging and then investigate log file or send the to Customer Support. Default: unchecked
Enable SNMP monitoring	Used by the Avalanche API calls. The SNMP results appear in the **Data Files** field. Default: unchecked **To configure SNMP monitoring:** 1. On the **Run** tab, check **Enable SNMP monitoring**. 2. Click **Configure** and then configure SNMP.
SNMP config file	Required if **Enable SNMP monitoring** is checked. Location of the SNMP config file (XML format). For example: **./files/snmp/snmp_config.xml**

Advanced > License

Hardware is an appliance	If unchecked, iTest makes the **av::connect [IP address] –type** STC call If checked, iTest makes the **av::connect [IP address] –type Appliance** STC call and you must specify a value for the **License file** property. Default: unchecked
License file	Required only if **Hardware is an appliance** is checked, Location of the Avalanche license file (XML format). For example: **./files/license/License.xml** **Note** If you specify Demo mode, then the license information is not required. See "Demo Mode" on page 1038 for details.

Demo Mode

See "Running a test using Demo mode" on page 1007

Real-time Stats Filter

During test execution, the server and client clusters return real-time statistics that depend on the test type. The full statistics table is quite large, and you can use the **Real-time Stats Filter** properties to limit the data that iTest captures and displays in the Console view.

Client real-time stats filter	Optional. Specify a search string. You can use the * wildcard character. iTest then captures and displays only the client data with matching text. In the example, we applied the following filter text: ***timeElapsed* *http***
Server real-time stats filter	Optional. Specify a search string. You can use the * wildcard character. iTest then captures and displays only the server data with matching text. In the example, we applied the following filter text: ***timeElapsed* *tcpConn***

Example filtered real-time statistics from the Console view (for a start action):

```
…..
Creating configuration array...
Generating test...

WARNING: Source file does not exist: ./files/certs/server/server.pem
WARNING: Source file does not exist: ./files/certs/server/server.pem
WARNING: Source file does not exist: ./files/certs/server/server.pem

WARNING: Source file does not exist: ./files/certs/server/server.pem
Stopping client cluster
Waiting for client cluster to complete
Stopping server cluster
Waiting for server cluster to complete
Removing test from client cluster
Removing test from server cluster
Uploading tests to client and server clusters
Starting test...
Client Statistics:
timeElapsed           | 4
```

```
http,attemptedTxns       | 0
http,successfulTxnsPerSec   | 0
http,txnsAwaitingResponse   | 0
http,abortedTxnsPerSec     | 0
http,successfulTxns      | 0
http,unsuccessfulTxns     | 0
http,abortedTxns        | 0
http,unsuccessfulTxnsPerSec  | 0
http,attemptedTxnsPerSec   | 0

Server Statistics:
timeElapsed           | 24
tcpConn,openConns       | 3
tcpConn,closedWithReset   | 10447
tcpConn,maxOpenConns     | 3
tcpConn,connsPerSec      | 2612
tcpConn,averageConnsPerSec  | 2612
tcpConn,closedWithError   | 0
tcpConn,closedWithNoError  | 0
```

```
Client Statistics:
timeElapsed           | 12
http,attemptedTxns       | 101366
http,successfulTxnsPerSec   | 17798
http,txnsAwaitingResponse   | 0
http,abortedTxnsPerSec     | 0
http,successfulTxns       | 101360
http,unsuccessfulTxns      | 0
http,abortedTxns         | 0
http,unsuccessfulTxnsPerSec  | 0
http,attemptedTxnsPerSec    | 17799

Server Statistics:
timeElapsed           | 32
tcpConn,openConns       | 9
tcpConn,closedWithReset    | 93173
tcpConn,maxOpenConns      | 9
tcpConn,connsPerSec      | 12748
tcpConn,averageConnsPerSec  | 7765
tcpConn,closedWithError    | 0
tcpConn,closedWithNoError   | 0
…
```

Tcl Interpreter

Avalanche sessions rely on a Tcl interpreter to do their work. You can specify the path to the correct interpreter here, in the session profile. Refer to Spirent Avalanche documentation for additional Tcl version information.

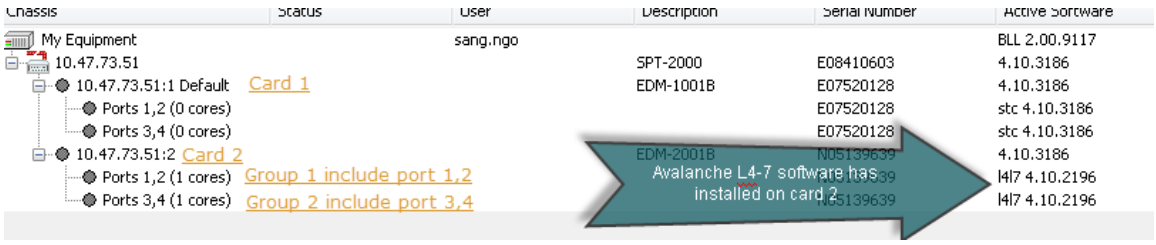
iTest requirements:

The Tcl interpreter must be **ActiveTcl** version **8.4.13** or later

Path to Tcl executable	**Required**. Type the directory path to the Tcl interpreter executable. Default: <None>
Avalanche Tcl API directory	**Optional**. Type the directory path to the Avalanche Tcl API library. If no value is specified, then iTest uses the latest API version in the registry. Default: <None>

Specifying cards, slots, port groups, and ports/virtual ports

A TestCenter chassis can have one or more ***cards*** (also called ***slots***). One or more port groups belong to each card and two ***ports*** belong to each ***port group***.



Appliance chassis have only one card, many ports belong to this example card:

Usable port

A ***usable*** port is an online or active port, and belongs to either server or client cluster units.

Examples

- Port in the **Port Provision List** option: Card 2, port 1
- **Server cluster units**: 2,1;0 (Card 2, group 1, unit 0)
- Chassis 10.47.73.52

You check on Avalanche Commander and see that Card 2, port 1 is in group 1. So the port you defined belongs to the Server Cluster Unit and is a usable port.

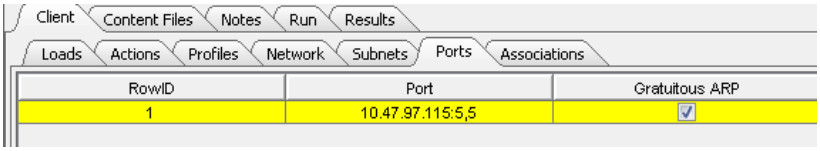
Count of ports

The number of usable ports must equal the number of ports in the config.tcl file.

```
TestName    {Test_demo}
TestType    {deviceComplex}
Ports       {10.47.73.51/2/1 10.47.73.51/2/3}
IsCompact   0
```

Client and Server cluster units

The cluster units options define which ports are used for the server and client interface. While creating a test using Avalanche Commander, for the server and client, click the **Ports** tab and specify ports for the test.



Here is the corresponding entry in the resulting config.tcl file:

```
av::config $tests_handle -topology.interface(1).dnsType {0}
av::config $tests_handle -topology.interface(1).gratuitousARP {on}
av::config $tests_handle -topology.interface(1).port {10.47.73.51/5/5}
av::config $tests_handle -topology.interface(1).primaryNameServer {}
av::config $tests_handle -topology.interface(1).secondaryNameServer {}
av::config $tests_handle -topology.interface(1).side {client}
```

Examples of invalid port organization

- Using one or more port groups for both Client and Server Cluster Unit, for example:
  - Server Cluster Units: **2,1;0 2,3;0** and Client Cluster Units: **2,2;**0 **2,1;0** (STC)
  - Server Cluster Units: **2 3** and Client Cluster Units: **1 3** (Appliance)

- Port provision List not belonging to Server or Client Cluster Unit:

  Server Cluster Units: 2,1;0 and Client Cluster Units: 2,2;0

  Port Provision List:

  - Port 1: Card 2, Port 1 -> group 1 belongs to Server Cluster Units
  - Port 2: Card 1, Port **2:** invalid, not belong to both Server and Client Units

- Number of usable ports not equal to number of ports in config.tcl file.

  In config.tcl file: Ports {10.47.73.51/2/1 10.47.73.51/2/3} (2 ports)

  However, in **Port Provision List** (Server Cluster Units: 2,1;0 and Client Cluster Units: 2,2;0):

  - Port 1: Card 2 port 1
  - Port 2: Card 1 port 2 -> **Unusable**

CHAPTER 69

Spirent Avalanche Commander NTAF sessions (Obsolete and Deprecated)

Important
The Spirent Avalanche Commander NTAF sessions are ***obsolete and is no longer supported*.**

This chapter is provided only to support existing implementations.

Note Avalanche is supported on Microsoft Windows only and is not supported on Linux.

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Overview of an Avalanche NTAF test

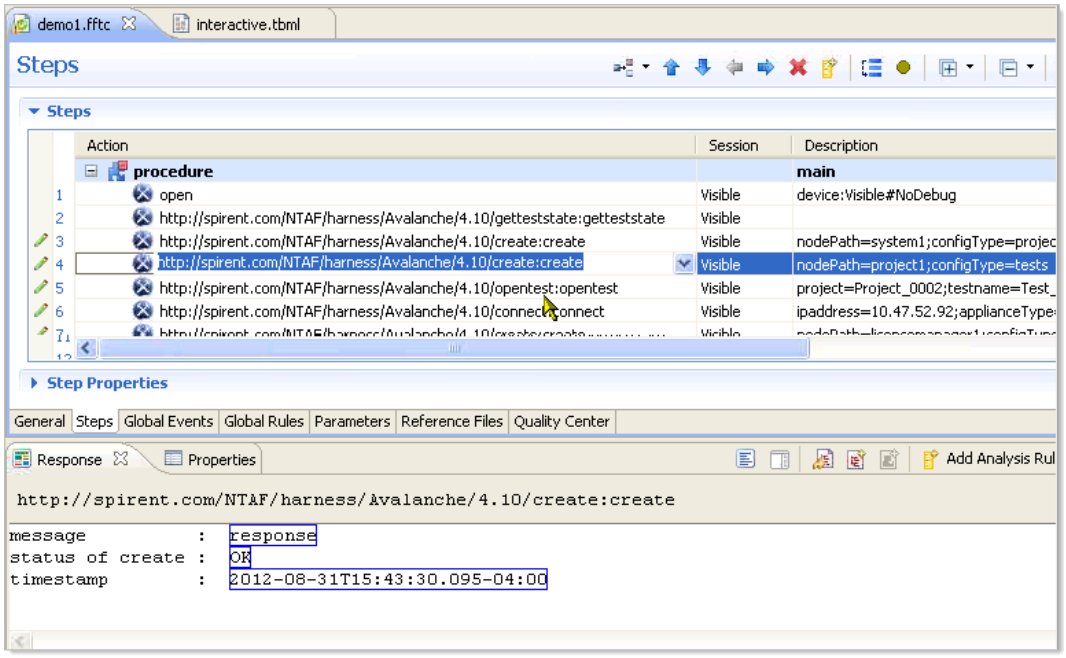
1. In iTest, when you start an Avalanche session, iTest launches the Avalanche user interface running on the Avalanche device.

   Now you can interact with Avalanche in the normal way. For example, you might set up a project, a test, a device to test, add an appliance, add a license for the device, create a port and an association for both the client and the server, run the test, and then close the Avalanche session window.

   While you perform the actions in Avalanche, iTest captures your actions. In addition, iTest captures the responses returned by Avalanche for each action. For example, the test submits a **getteststate** action, Avalanche returns a response, and both the command and the response are captured by iTest. (As usual, you can also view the captured items in the iTest Capture view.)

2. You can now save the captured steps and responses as a iTest test case. As needed, you can modify and update the test case. For example, delete the **create project** and **create license manager** steps because they will not be needed in future runs. You might also add an

analysis rule for the **getteststate** step that waits until the text “TEST COMPLETED” is returned in the response before proceeding with the next step in the test case.



You can execute the test case at any time. When the iTest test case executes, iTest starts an Avalanche session and executes the steps exactly as you performed them during your manual session.

About Avalanche data

- iTest does not use the Spirent Avalanche session steps to analyze the traffic data—other steps in your iTest test case can do that.
- iTest saves all Avalanche responses as structured data. iTest auto-maps responses and auto-generates queries for response data (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create Avalanche response maps and can immediately write analysis rules that use the auto-generated queries. See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, “Queries view” on page 352, and “Structure view” on page 358.

Avalanche action reference

The name of each iTest action is the same as the name of the Avalanche API function that it executes. See the *Avalanche Tcl API Reference* for details on operation and usage.

If the Avalanche Tcl command associated with the iTest step/action requires arguments, then, for the selected step in the Test Case editor, you can view and set the argument values in the **Step Properties**.

Each iTest action returns the same values as the corresponding Avalanche command.

Non-standard commands

Action performed in Avalanche	Action captured in iTest
reserve multiple ports	connect

Starting a session with Avalanche

The process described in this section is covered in the video on Avalanche at: http://www.fanfaresoftware.com/videos

Step 1 Log in to the NTAF server

Follow the instructions in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

Step 2 Define a Spirent Avalanche NTAF session

Configure an Avalanche NTAF session profile as described in “Configuring a session for Avalanche NTAF” on page 1304.

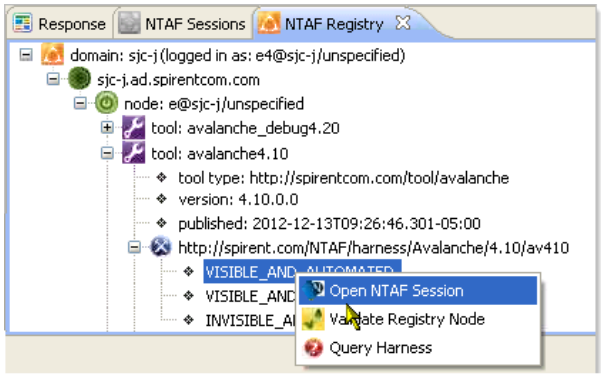
Step 3 Start a session with Avalanche

Use one of the following methods to start an Avalanche session in iTest.

- In either the **Sessions** section on the **Connect to Devices** page or the Session Profile editor, open the session profile that you configured in Step 2 “Define a Spirent

Avalanche NTAF session” on page 1302 and then click **Start**. This is the typical method for manual testing.

- Start the session from the NTAF Registry view: Right-click the **Visible and Interactive** execution mode in the tree and select **Open NTAF Session**. (The **Visible and Automated** and **Invisible and Automated** modes are used during iTestRT automated execution.)



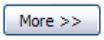
For a more detailed description of the Registry view, see “NTAF Registry view” on page 1312 in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

Note To allow maximum flexibility and portability, iTest uses the NTAF standard to communicate with the Avalanche session. As a result, you often see the term “NTAF session” associated with Avalanche sessions in iTest.

Configuring a session for Avalanche NTAF

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Session Properties, Spirent Avalanche Commander

NTAF node	Specify the URL or hostname of the XMPP server (the “NTAF server”).
NTAF tool	Specify the version of the Avalanche NTAF provider to use.

Setting preferences for NTAF sessions

All NTAF preference settings are used by default whenever a user attempts to connect iTest or the NTAF Proxy service to the NTAF server. The settings are applied to all NTAF-compliant providers — not only to the particular type of NTAF session that you are currently working on.

See “Setting preferences for NTAF sessions” on page 1311 in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

CHAPTER 53

Working with NTAF sessions in Spirent iTest

Important

The Spirent TestCenter NTAF and Spirent Avalanche Commander NTAF sessions are ***obsolete and is no longer supported***.

Any references is provided only to support existing implementations.

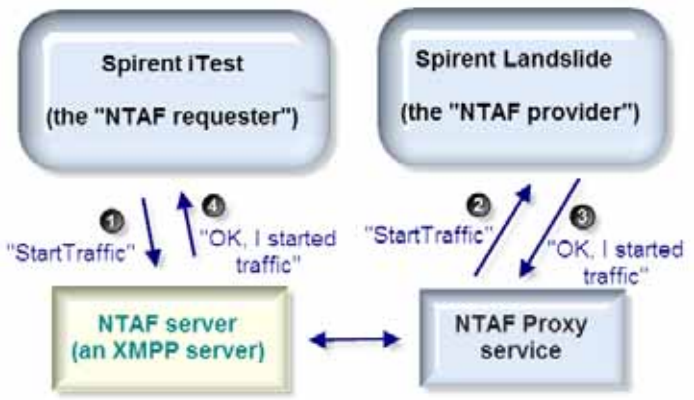
This chapter provides reference information for all session types that make use of the NTAF standard in enabling communications between the test equipment and Spirent iTest. Information on working in the Spirent iTest sessions appears in the appropriate chapters.

How Spirent iTest works with NTAF-enabled devices and applications

The NTAF server is an XMPP server that is installed and managed by your IT administrator.

The NTAF server handles communications between the session in iTest (the NTAF requester) and the test equipment (the NTAF provider) by means of the NTAF Proxy service.

In this example, a Spirent Landslide NTAF session is communicating with the test equipment (Spirent Landslide). In a typical installation, all of the elements are installed on the same host.



To work with NTAF-enabled sessions on Spirent iTest

Step 1 Start the NTAF server

Your IT administrator starts the NTAF server.

Step 2 Log Spirent iTest in as a client on the NTAF server

1 In Spirent iTest, on the **NTAF Registry** view (**Window > Show View > Other > NTAF > NTAF Registry**), click  to connect to the server.

To support troubleshooting, the NTAF Registry view lists each host that is running an NTAF tool (NTAF provider) and displays information about each tool. Details at “NTAF Registry view” on page 991.

2 On the on the **Login to NTAF Server** dialog box, specify the following settings:

Tip You can make these settings the default. See “Setting preferences for NTAF sessions” on page 990.

Login settings

Property Name	Equivalent iTestRT switch	Description
NTAF server	**--ntaf.server <URI>**	Specify the IP address or hostname of the NTAF server (provided by your IT administrator).
Username / Password	**--ntaf.user <value>** **--ntaf.password <value>**	Specify the XMPP credentials to use to enable iTest to access the NTAF server as a client. **IMPORTANT** The credentials represent iTest as a client on the NTAF server. Remember that the Proxy service is a different client on the NTAF server and therefore has a different username. **Note** You can set the authentication information in the **NTAF** preferences page in iTest. The values are not used when an application login page asks for credentials
Automatically sign me in **Prompt for login every time** **Do not auto-login**	**(none)**	Specify what should happen when you open a session with the NTAF server: Apply the specified credentials or open this dialog box and request credentials.
Remember login settings	**(none)**	Check the box to use the specified credentials by default when logging in.

Advanced settings

Property Name	Equivalent iTestRT switch	Description
Domain	**--domain <URI>**	XMPP domain name. Default: The NTAF server hostname. When you log onto an XMPP server, you get an ID (called a Jabber ID, JID) like username@ntafxmpp.spirent.com/unspecified “username” is the user, "ntafxmpp.spirent.com" is the domain, which is usually the same as the value of the **NTAF server**. However XMPP servers can be configured so that domain and server are different. (In the example, “unspecified” is the **Resource**).
Server port	**--ntaf.port <value>**	Port address of NTAF server. Default: 5222
Login name	**--login <name>**	XMPP login name. Default: The value of the **Username** credential used to log in.
Resource	**--resource <value>**	XMPP resource. The last part of the JID is the resource. For example, “unspecified” in **username@ntafxmpp.spirent.com/unspecified** Default: “unspecified”
Registry address	**--regAddress <value>**	Pubsub server address on NTAF server. Default: “pubsub.” followed by the XMPP domain.
Registry root node	**--regRoot <value>**	Pubsub root node for the NTAF registry. All NTAF provider registration information is stored under the root node. Default: **ntaf.tools** **Note** The default setting is typically correct. While the NTAF standard allows for other names, changing names may confuse the providers that you are trying to communicate with.
Do not try in-band registration if login fails	**--inband <value>**	In-band registration means that the NTAF server will register a new account for your username if it does not yet exist. If you check the box, then your username must exist on the NTAF server for you to log in.
Do not reconnect if connection is lost	**--reconnect <value>**	Check the box to not automatically retry XMPP connections if and when the connection with the server fails.
Open window showing XMPP packets	**(none)**	Check the box to open a window showing XMPP packets sent between iTest and the NTAF server.
Allow NTAF views to start sessions and send requests	**(none)**	Enable the menus in the NTAF Sessions view that start sessions and send requests. See “Starting the NTAF Proxy service from the command line” on page 993.

Step 3 Start the NTAF Proxy service and log it in as a client on the NTAF server

You launch the NTAF Proxy service to enable installed Spirent tools (Avalanche or Landslide) or other NTAF-enabled tools to communicate with the NTAF server.

To support troubleshooting, the NTAF Proxy view displays information about each NTAF tool (NTAF provider) running on the local host. Details at “NTAF Proxy view” on page 992.

◆ To launch the Proxy service and connect it to the NTAF server

1 On the **NTAF Proxy** view (**Window > Show View > Other > NTAF > NTAF Proxy**), click  to connect to the server.

While it is not recommended, you have the option to start the NTAF Proxy using a command-line interface. For details, see your IT admin and “Starting the NTAF Proxy service from the command line” on page 993.

2 On the on the **Login to NTAF Server** dialog box, specify the following settings:

Tip You can configure auto-login and default settings for the Proxy service. See “Setting preferences for NTAF sessions” on page 990.

Login settings

NTAF server	Specify the IP address or hostname of the NTAF server (provided by your IT administrator).
User / Password	Specify the XMPP credentials to use to enable the NTAF Proxy service to access the NTAF server as a client. **IMPORTANT** The credentials represent the Proxy service as a client on the NTAF server. Remember that iTest is a different client on the NTAF server and therefore has a different username. **Note** You can set the authentication information in the **NTAF** preferences page in iTest. The values are not used when an application login page asks for credentials.
Automatically sign me in **Prompt for login every time** **Do not auto-login**	Specify what should happen when you open a session with the NTAF server: Apply the specified credentials or open this dialog box and request credentials.

Advanced settings

Domain	XMPP domain name. Default: The NTAF server hostname. When you log onto an XMPP server, you get an ID (called a Jabber ID, JID) like username@ntafxmpp.spirent.com/unspecified “username” is the user, "ntafxmpp.spirent.com" is the domain, which is usually the same as the value of the **NTAF server**. However XMPP servers can be configured so that domain and server are different. (In the example, “unspecified” is the **Resource**).
Server port	Port address of NTAF server. Default: 5222

Login name	XMPP login name. Default: The value of the **User** credential used to log in.
Resource	XMPP resource. The last part of the JID is the resource. For example, “unspecified” in **username@ntafxmpp.spirent.com/unspecified** Default: “unspecified”
Registry address	Pubsub server address on NTAF server. Default: “pubsub.” followed by the XMPP domain.
Registry root node	Pubsub root node for the NTAF registry. All NTAF provider registration information is stored under the root node. Default: **ntaf.tools** **Note** The default setting is typically correct. While the NTAF standard allows for other names, changing names may confuse providers that you are trying to communicate with.
Do not try in-band registration if login fails	In-band registration means that the NTAF server will register a new account for your username if it does not yet exist. If you check the box, then your username must exist on the NTAF server for you to log in.
Do not reconnect if connection is lost	Check the box to not automatically retry XMPP connections if and when the connection with the server fails.
Open window showing XMPP packets	Check the box to open a window showing XMPP packets sent between iTest and the NTAF server.

Step 4 Spirent iTest is now ready to open the NTAF-enabled session.

Setting preferences for NTAF sessions

To view or edit preferences, click **Window > Preferences**.

Important The settings that you specify here are used by default whenever a user attempts to log in to the NTAF server and are applied to all NTAF-compliant providers, not only to the particular type of NTAF session that you are currently working on.

Preferences for logging iTest in to the NTAF server

On the **Preferences** page, click **NTAF**

The settings that you can set on the **Preferences** page are exactly the same as on the NTAF server **Login** dialog box, as described in “To work with NTAF-enabled sessions on Spirent iTest” on page 986.

Preferences for logging the NTAF Proxy service in to the NTAF server

On the **Preferences** page, click **NTAF > Proxy**

The settings that you can set on the **Preferences** page are exactly the same as on the NTAF server **Login** dialog box, as described in “Start the NTAF Proxy service and log it in as a client on the NTAF server” on page 988.

The NTAF perspective

Click  and select **NTAF** to switch to the NTAF perspective and display the following views:

- The **NTAF Registry view**
- The **NTAF Proxy view**

NTAF Registry view

The NTAF Registry view displays each host that is running an NTAF tool.

The tools are listed in the NTAF registry on the XMPP server, which is a pubsub node with one child node for each registered NTAF tool. iTest groups nodes by host. The XMPP NTAF registry, however, has no such hierarchy.

Registered nodes can be inactive (red) or active (green). A node can refer to one tool (standalone tool) or many tools (proxy). The **spirent_proxy** node is the communication medium for Spirent tools such as Spirent Landslide.

To open the NTAF view

- Click **Window > Show View > Other > NTAF > NTAF Registry**
- The view is also part of the **NTAF** perspective. Click the perspective button and select **NTAF**.

Refresh the view
(disabled if any tool is active)

- Clear all NTAF tools from memory and from the view
- Read tools from disk
- Update the view with the current tool information

Log in to / out of NTAF server
(logout is disabled if any NTAF tool is active)

NTAF server with registry

Host information

NTAF entity node
-- **spirent_proxy** in this example is connected (indicated by the green icon)

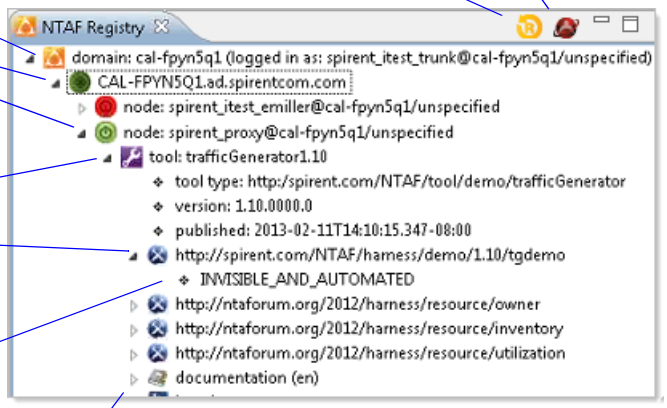
Tool leaf node. Standalone tools have one leaf node, proxies can have many

NTAF info for the tool

Harness info for the tool. There can be many harnesses. Harnesses define the interface for the tool — the actions and parameters that the tool supports

Modes supported by the harness
NTAF defines three tool modes:

- visible_and_interactive — (running under iTest
- visible_and_automated — running under iTest
- invisible_and_automated — running under iTestRT



documentation (and children) — Info about the tool

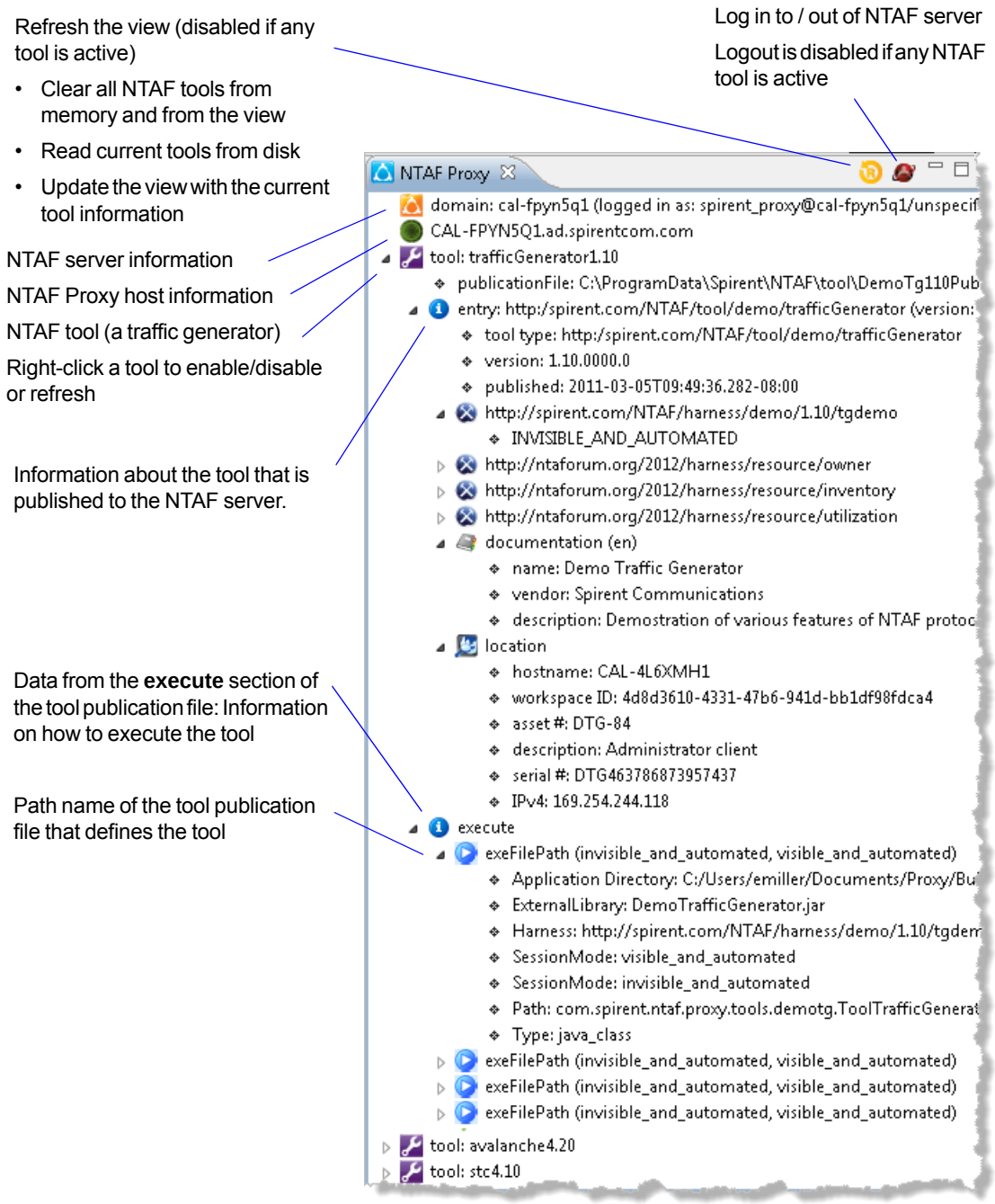
location (and children — Info about the host that the tool is running on

NTAF Proxy view

The NTAF Proxy view displays information on NTAF tools (NTAF providers) running on the local host.

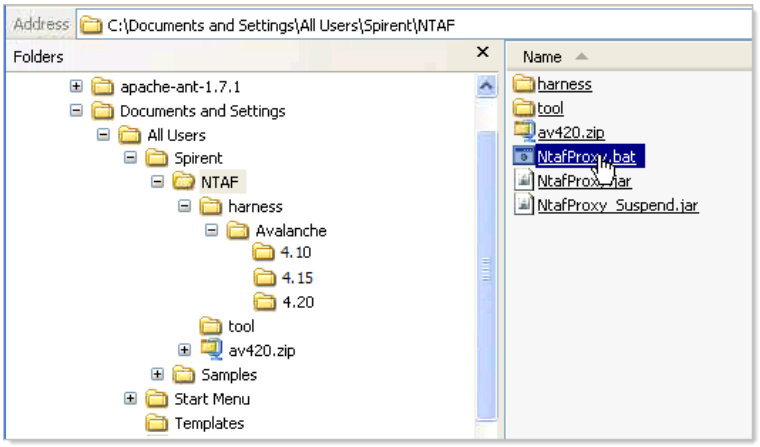
To open the NTAF Proxy view

- Click **Window > Show View > Other > NTAF > NTAF Proxy**
- The view is also part of the **NTAF** perspective. Click the perspective button and select **NTAF**.



Starting the NTAF Proxy service from the command line

The NTAF installation process creates the following directory structure under **Documents and Settings**. Run **NtafProxy.bat** to start the NTAF proxy.



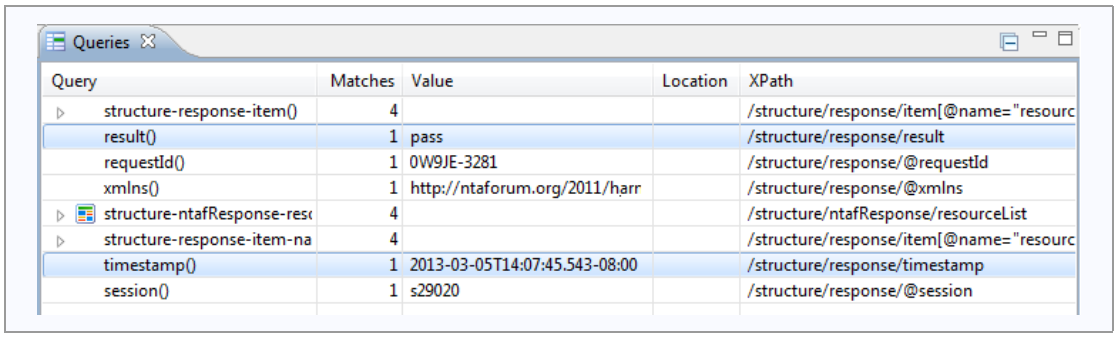
Automating NTAF test cases

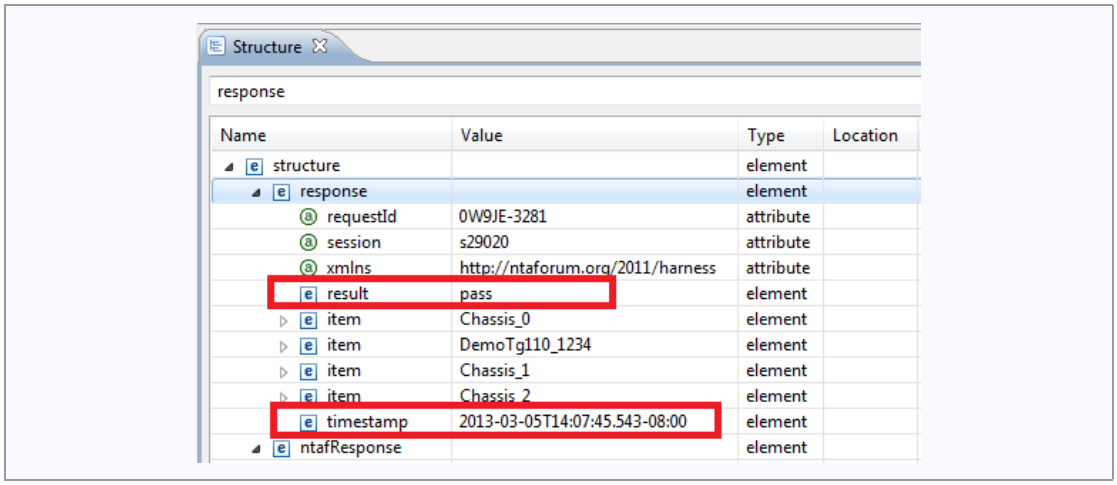
To execute an NTAF test case using iTestRT (iTest Runtime), connect to the NTAF server using the NTAF options. See “iTestRT command reference” on page 709.

NTAF Response Data in iTest

NTAF Metadata

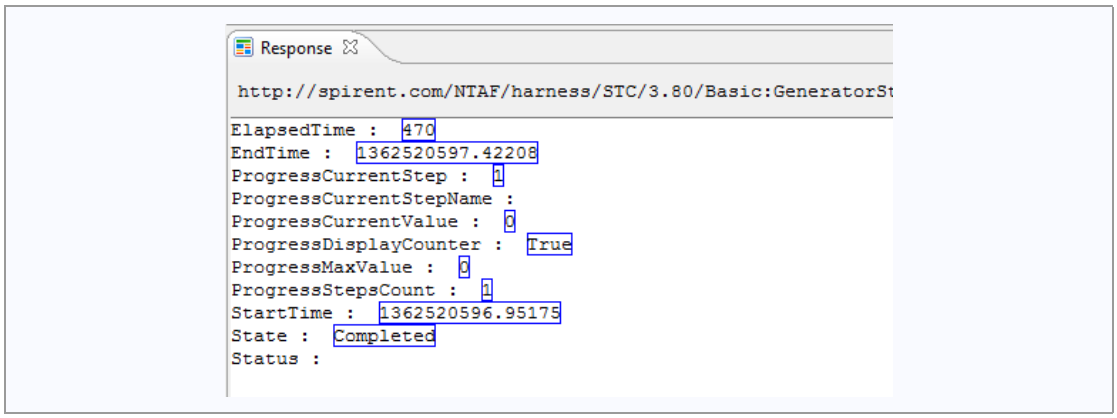
NTAF has common fields like **result** and **timestamp**. As with the metadata of other session types, common NTAF data fields do not appear in the response text. The data is available, however, as a query in the **Queries** view and in the **Structure** view:





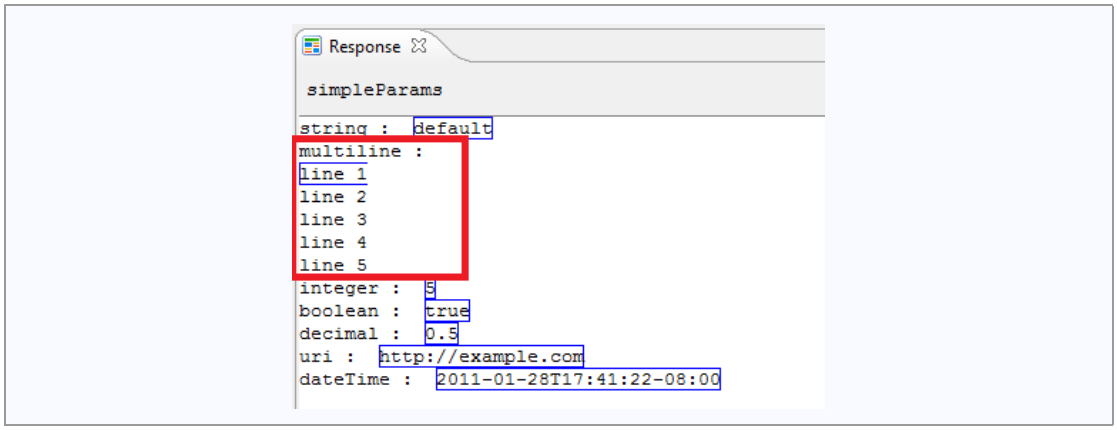
Response items as name-value pairs

Simple NTAF response item data appears as name-value pairs in the **Response** view.



Multi-line data

To help you to generate meaningful response maps for multi-line data, the data item name appears on one line followed by the data:



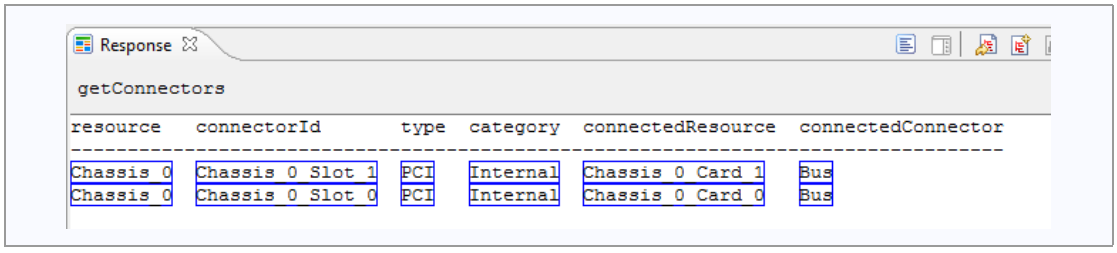
Tables

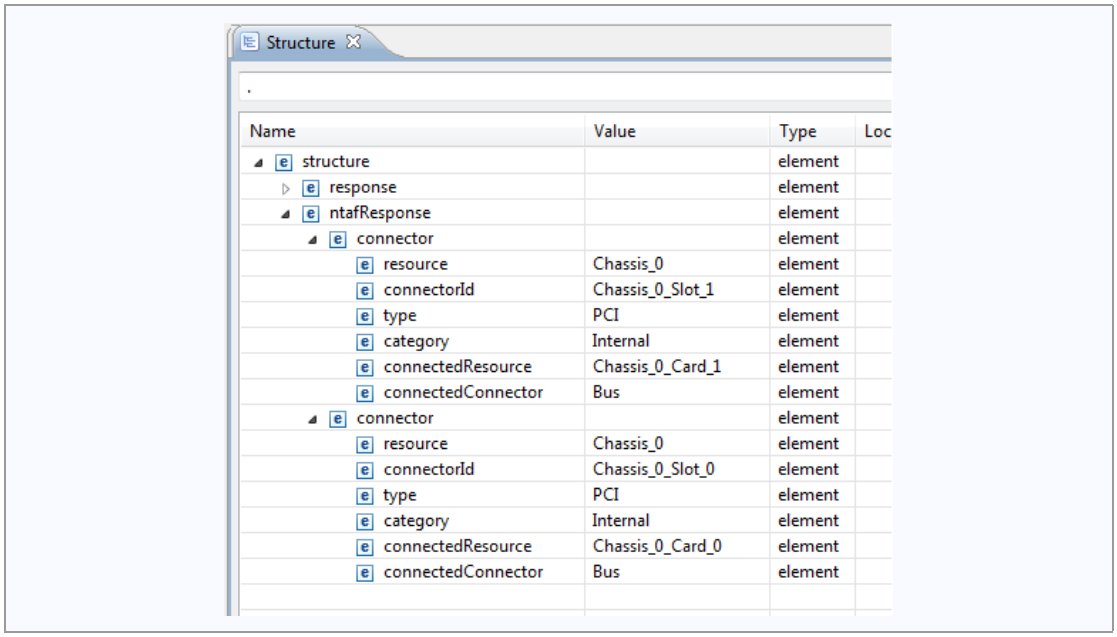
NTAF data that is structured as a table can have two different types of structure:

- **Multi-valued group**

Each group contains the same set of items (attributes). In the table representation that is displayed in the **Response** view, the names of the items are the column headers and the item values appear as a row in the table.

In our example, the NTAF table data is contained two response groups. The items **resource**, **connectorId**, **type**, **category**, **connectedResource**, and **connectedConnector** appear in each response group. The **Structure** view shows the internal data structure..

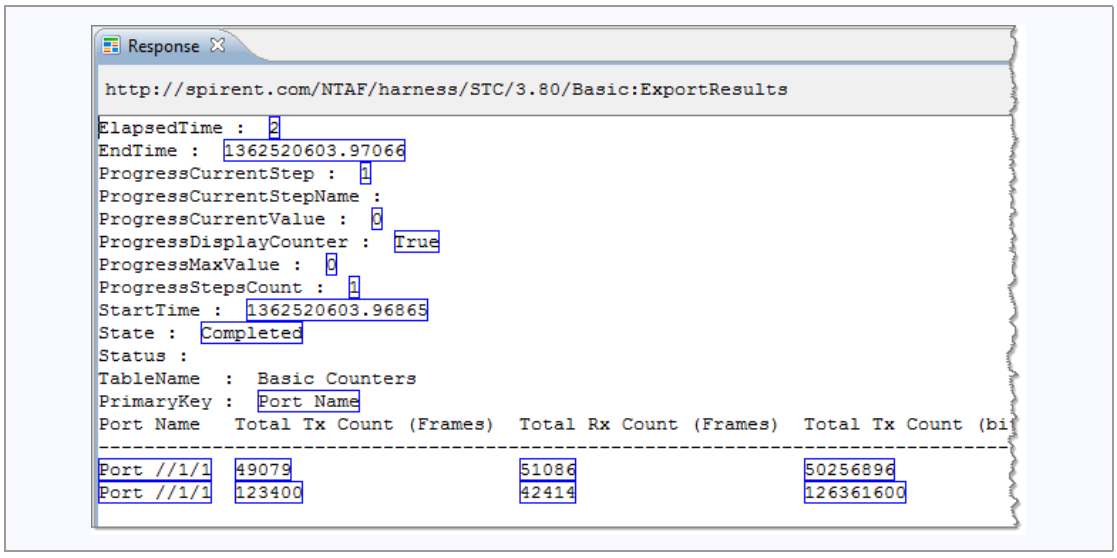


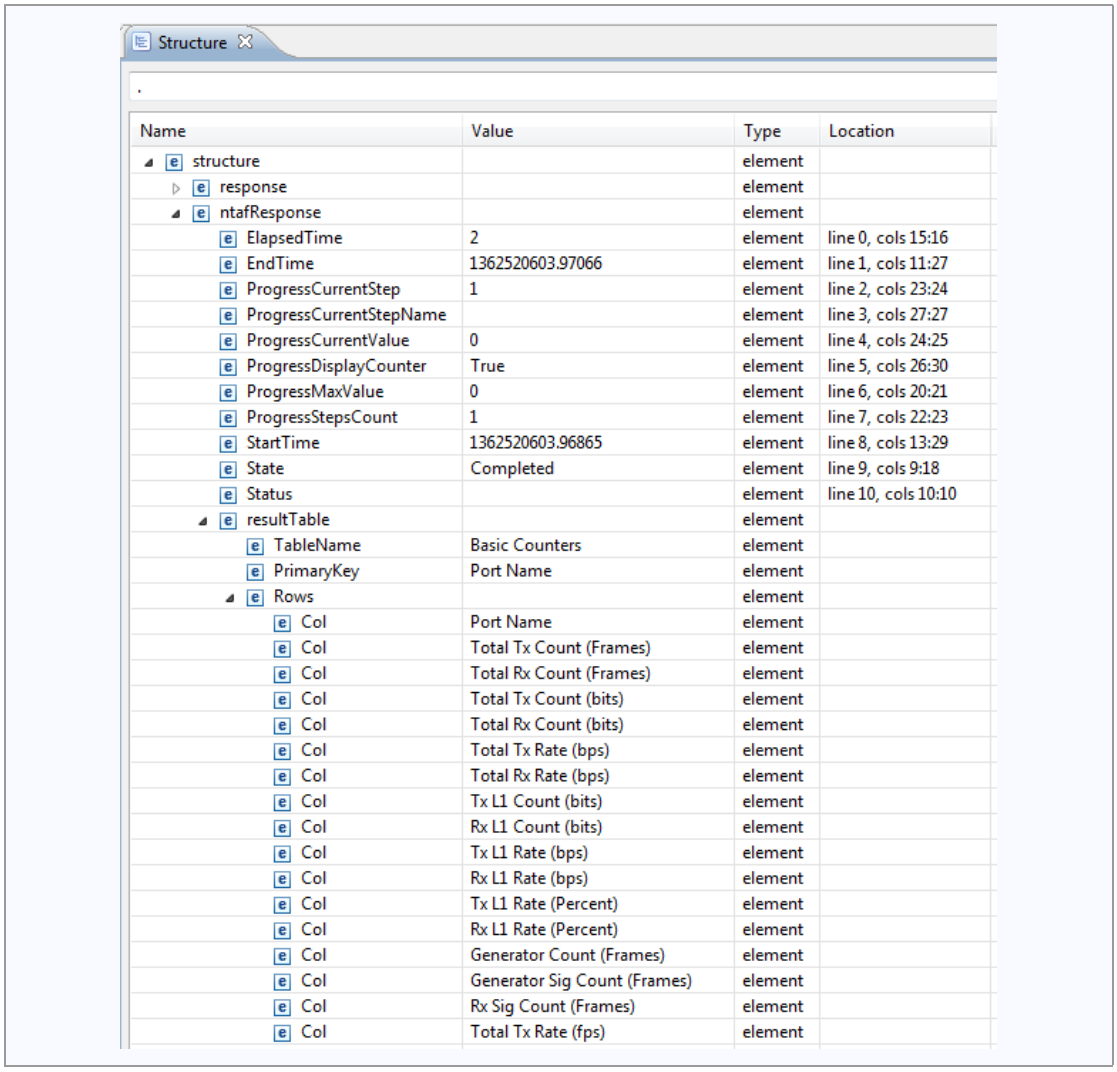


- **Multi-valued response group with one multi-value item**

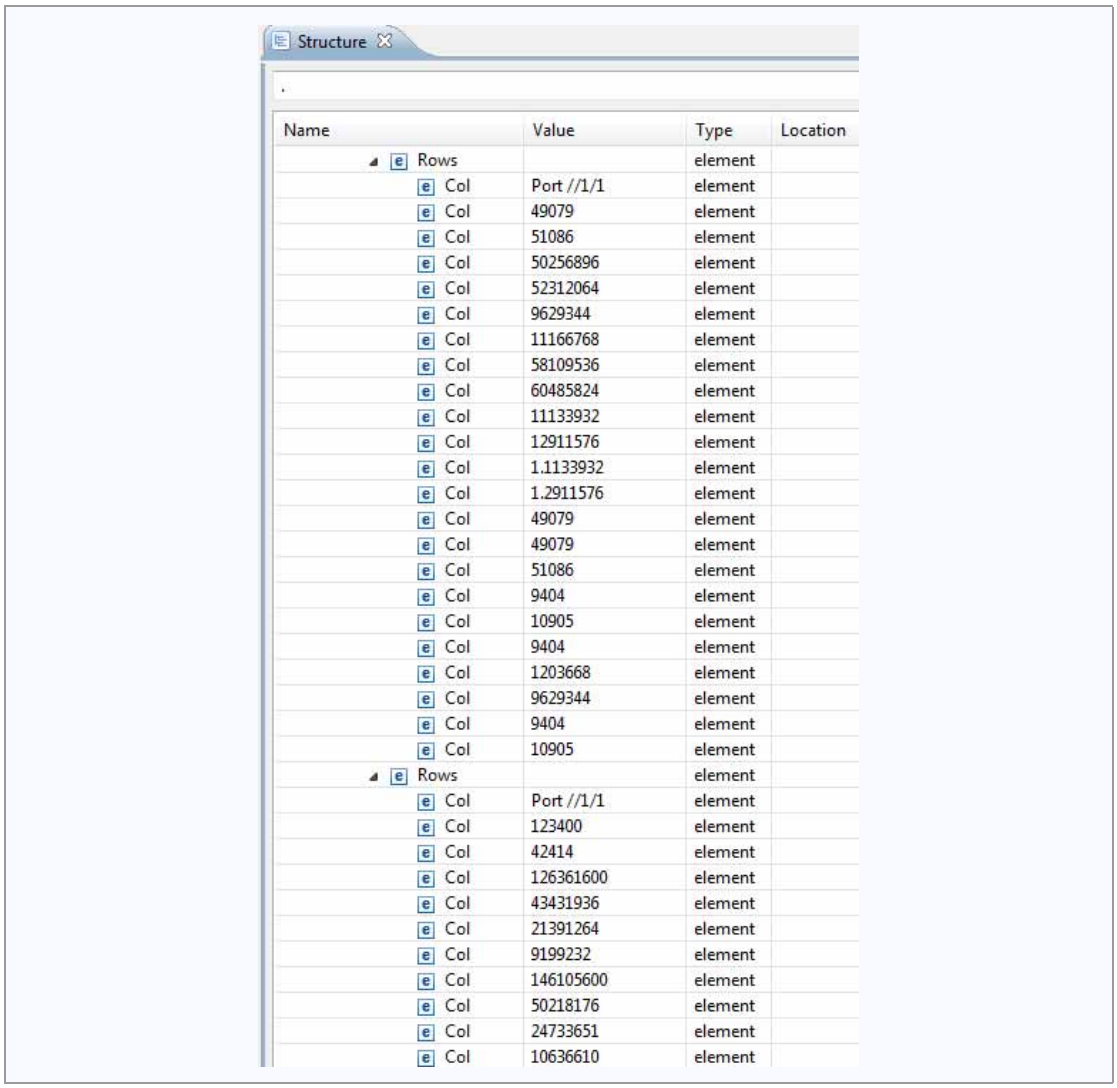
  You can see that this type of table has the same structure as the first type of table. Notice also that the response includes name-value data in addition to the table data.

  In this example, the NTAF table data contains three response groups — each with a multi-valued item. The first group provides the column names and contains values **Port Name**, **Total Tx Count (Frames)**, **Total Rx Count (Frames)**, and so on. Subsequent group item values define the rows. The second group contains values **Port //1/1**, **49079**, **42414**, and so on. The third group contains values **Port //1/1/**, **123400**, **42414**, and so on. The **Structure** view shows the internal data structure.




Structure
.
Name
Value
Type
Location
structure
element
response
element
ntafResponse
element
ElapsedTime
2
element
line 0, cols 15:16
EndTime
1362520603.97066
element
line 1, cols 11:27
ProgressCurrentStep
1
element
line 2, cols 23:24
ProgressCurrentStepName
element
line 3, cols 27:27
ProgressCurrentValue
0
element
line 4, cols 24:25
ProgressDisplayCounter
True
element
line 5, cols 26:30
ProgressMaxValue
0
element
line 6, cols 20:21
ProgressStepsCount
1
element
line 7, cols 22:23
StartTime
1362520603.96865
element
line 8, cols 13:29
State
Completed
element
line 9, cols 9:18
Status
element
line 10, cols 10:10
resultTable
element
TableName
Basic Counters
element
PrimaryKey
Port Name
element
Rows
element
Col
Port Name
element
Col
Total Tx Count (Frames)
element
Col
Total Rx Count (Frames)
element
Col
Total Tx Count (bits)
element
Col
Total Rx Count (bits)
element
Col
Total Tx Rate (bps)
element
Col
Total Rx Rate (bps)
element
Col
Tx L1 Count (bits)
element
Col
Rx L1 Count (bits)
element
Col
Tx L1 Rate (bps)
element
Col
Rx L1 Rate (bps)
element
Col
Tx L1 Rate (Percent)
element
Col
Rx L1 Rate (Percent)
element
Col
Generator Count (Frames)
element
Col
Generator Sig Count (Frames)
element
Col
Rx Sig Count (Frames)
element
Col
Total Tx Rate (fps)
element

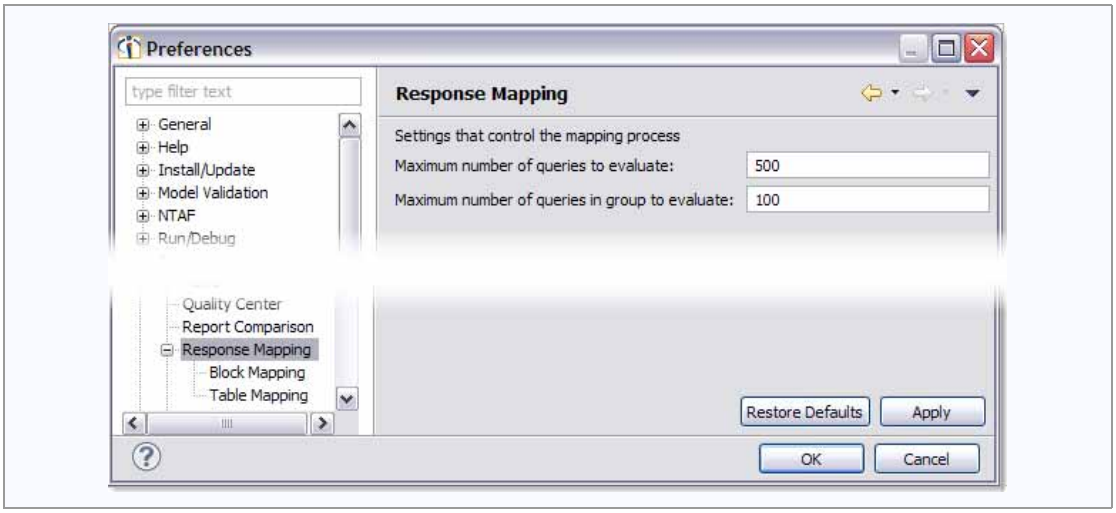
Here is some of the structure data for the rows of values.



Specifying the maximum number of queries to display

Sometimes NTAF tools can generate a lot of data, especially when tables of results are returned. iTest, by default, sets the maximum number of queries (“blue boxes”) that can be generated at 100. You have the option to set a preferences setting to map more (or fewer) queries. In iTest,

click **Windows > Preferences** and navigate to **Spirent > Response Mapping**. Set the value for the **Maximum number of queries to evaluate** property.



High Level Structure of NTAF response data

NTAF structure data has two major sections.

Note Spirent recommends that you use the **ntafResponse** structure for group and item data for the reasons given in this section.

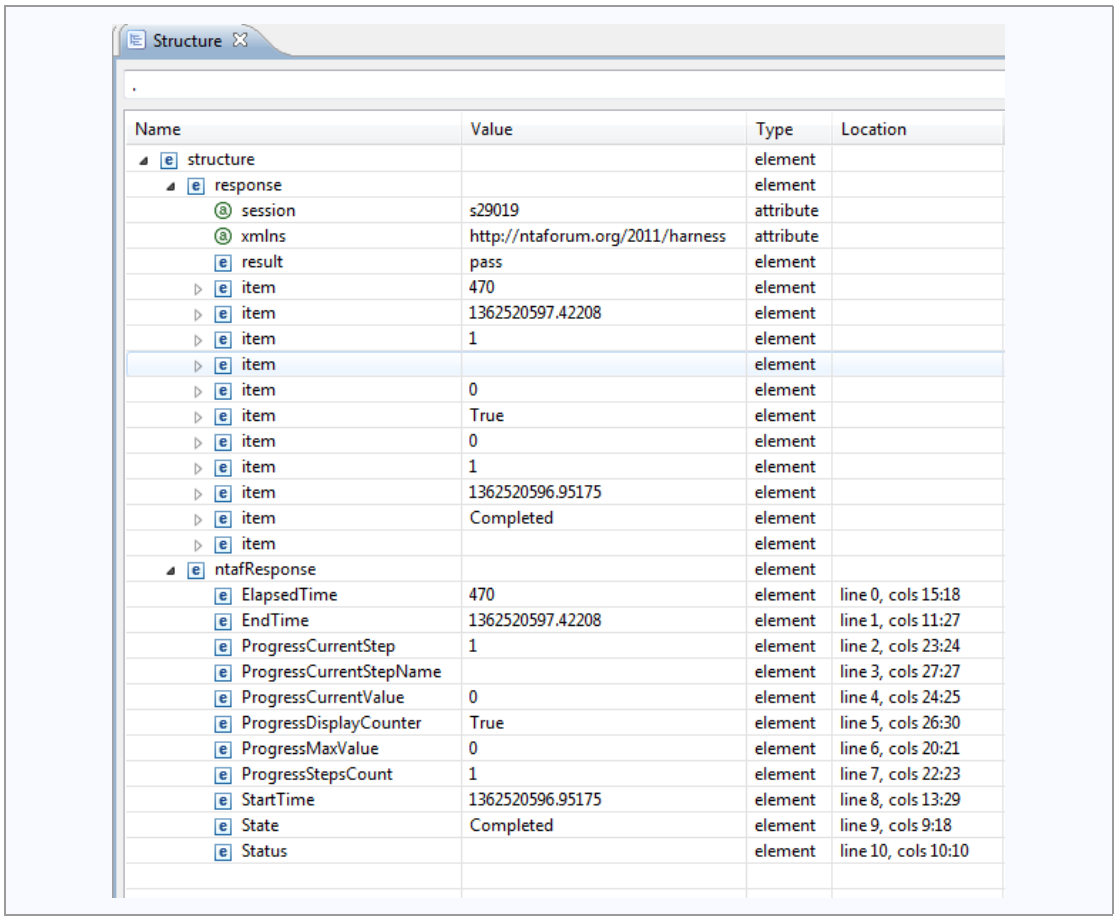
- **response**

  The **response** section was developed first and was available in the earliest versions of iTest that supported NTAF. It organizes data by XML element tag. While this strategy captures all the data, it can be hard to use because many tags in NTAF data are the same (like **item** in our example, or **group**). This structure is retained for compatibility with older versions.

- **ntafResponse**

  Later versions of iTest support the **ntafResponse** structure, which is based on group and item names. Notice how the elements of **ntafResponse** are named using the names contained in the

items. This strategy better corresponds to NTAF structure and is much easier to navigate. By default, iTest queries are designed to refer to the **ntafResponse** structure.



CHAPTER 71

Spirent Landslide NTAF sessions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Overview: Landslide session window

1 In iTest, when you start a Landslide session, iTest launches the Landslide TAS user interface running on the Landslide device.

   Now you can interact with the TAS in the normal way. For example, you might load a test configuration, start the test session, collect the responses, wait to collect several data sets, stop the test session, request the test session results, and then close the test session.

   While you perform the actions in the TAS, iTest captures your actions. In addition, iTest captures the responses returned by Landslide for each action. For example, you submit a **loadConfig** action, the TAS returns a response, and both the command and the response are captured by iTest. (As usual, you can also view the captured items in the iTest Capture view.)

   You can capture entire sets of Landslide property settings. In the resulting iTest test case, you can convert any property setting into a variable that is set dynamically at runtime. See “Capturing Landslide property settings” on page 1323.

2 When you are finished working on the TAS, you return to iTest and close the Landslide session window. iTest disconnects the Landslide session and closes the TAS user interface.

3 You can now save the captured steps and responses as a iTest test case. As needed, you can modify and update the test case (for example, modify the **ImportTestSuite** step by causing it to load a different Landslide test suite file or replace the filename with a variable whose value is set at runtime).

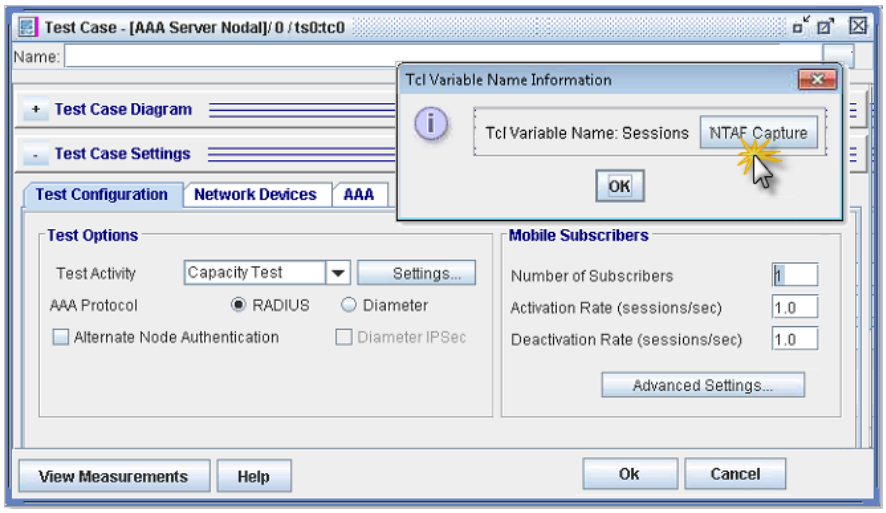
   You can execute the test case at any time. When the iTest test case executes, iTest starts a Landslide session and executes the steps exactly as you performed them during your manual session.

Note iTest supports multiple concurrent Landslide sessions.

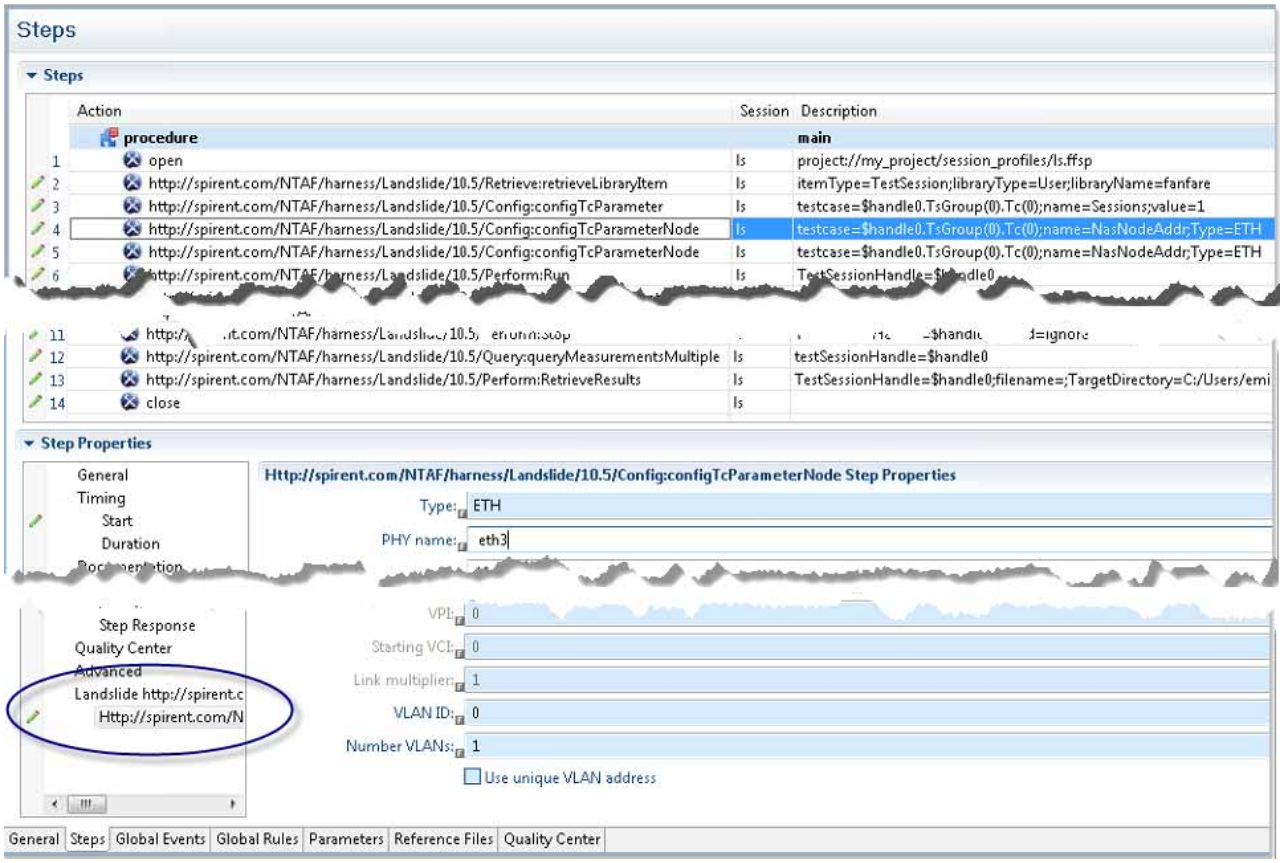
Capturing Landslide property settings

You can capture entire sets of Landslide property settings. For example, while viewing node settings on the Landslide **Test Case** page, press **F2** to open the expected dialog box — with the addition of an **NTAF Capture** button. Click the button to capture the node property settings.

Note The settings are not captured until you close the Landslide **Test Case** page (due to a Java Web Start security restriction).



The settings appear as step properties in the resulting iTest test case:



You can capture the following types of settings:

Setting Type	Captured Action in iTest Test Case	Description
Basic	configTcParameter	Name value pair
Node	configTcParameterNode	Set of parameters that make up the node
Array	configTcParameterArray	Array of basic parameters
SUT	configTcParameterSut	Data about a SUT

You can convert any property setting into a variable that is set dynamically at runtime.

Note The following commands are not supported in the current release of Landslide. These commands will support both Capture and Replay.

- CommandTestCase
- GeneratePerSessionReport
- CapturePortReservation
- PortCapture
- retrieveTestServerInfo

Landslide action reference

The name of each iTest action is the same as the name of the Landslide Tcl API function (ls:: action) that it executes. See the *Landslide Tcl API Object and Perform Function Reference* (on *Spirent Knowledge Base*) for details on operation and usage.

If the Landslide Tcl command associated with the iTest step/action requires arguments, then, for the selected step in the Test Case editor, you can view and set the argument values in the **Step Properties**.

Each iTest action returns the same values as the corresponding Landslide command.

Starting a session with Landslide

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Step 1 Log in to the NTAF server

Follow the instructions in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

Step 2 Define a Spirent Landslide NTAF session

Ensure that the Landslide session profile or device is properly configured. See “Session profile property settings for Landslide sessions” on page 1325.

Step 3 Start a session with Landslide

Note iTest supports multiple concurrent Landslide sessions.

Use either of the following methods to start a Landslide session in iTest.

- In the Topology view, select and start the session. This is the typical method for manual testing.
- In the Session Profile editor, open the session profile and click **Start**.

Note To allow maximum flexibility and portability, iTest uses the NTAF standard to communicate with the Landslide session. As a result, you often see the term “NTAF session” associated with Landslide sessions in iTest.

Session profile property settings for Landslide sessions

Harness

An NTAF harness defines the action set for a iTest session with an NTAF provider (like Spirent Test Center or Landslide).

As new provider versions are released, new versions of the associated NTAF harness may also be released to implement the latest commands and features. The harness version number is typically identical to the provider version number.

Harness URI	Harness version to use for submitting actions to a session. **Note** New harness versions are designed to operate properly for test cases built using older harnesses, as long as the newer version supports the old harness or the subharnesses used in the test case. The **Action** cell in the Test Case editor list all actions supported by the harness and its subharnesses. If an action from an older version is no longer supported by the new version, then an error appears for the affected step in the test case. Click the error icon to view a description. Typically, you can select the new version of the action to correct the error. The **Action** cell in the Test Case editor lists actions by their subharness followed by the action name. Old test cases that contain actions specified only by the action name display an error icon by the action. You must select the correct action in list by looking for the correct subharness/action. Once you select the correct subharness/action, the properties are auto-populated.

Important If you change the harness in a session profile, then review all property settings to ensure that the correct settings are used (changed values are highlighted in yellow).

In addition, if you change the harness version in session profile A, then all property settings in session profile B that inherit from A are reset to inherit by default, including values that had been manually specified in B to override the inherited setting. This means that you must edit B after the change to ensure that the override takes place.

For example, for sessions that inherit property settings from another session profile, you cannot change the inheriting harness version. To be able to change the harness version, you must create a session profile that does not inherit property settings.

Harness > Session Properties

NTAF node	Specify the URL or hostname of the XMPP server (the “NTAF server”).
NTAF tool	Specify the version of the Landslide NTAF tool (provider) to use.
Landslide host	Specify the URL of the Landslide page at which to begin. This is the information that you typically type into a browser's Address bar. If you do not specify a URL, then the browser opens without going to a page. You can type the URL into the address box. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**
User name/ Password	If the session will access a secure site, then provide the credentials that iTest should submit to gain access. • Internal browser: If you set the properties here, in the session profile, then before starting the session, the browser will not prompt you to enter the credentials. • Internet Explorer: You must specify HTTP credentials here, in the session profile. **Note** These values are HTTP credentials. The values are not used when an application login page asks for credentials.

Setting preferences for NTAF sessions

All NTAF preference settings are used by default whenever a user attempts to connect iTest or the NTAF Proxy service to the NTAF server. The settings are applied to all NTAF-compliant providers — not only to the particular type of NTAF session that you are currently working on.

See “Setting preferences for NTAF sessions” on page 1311 in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

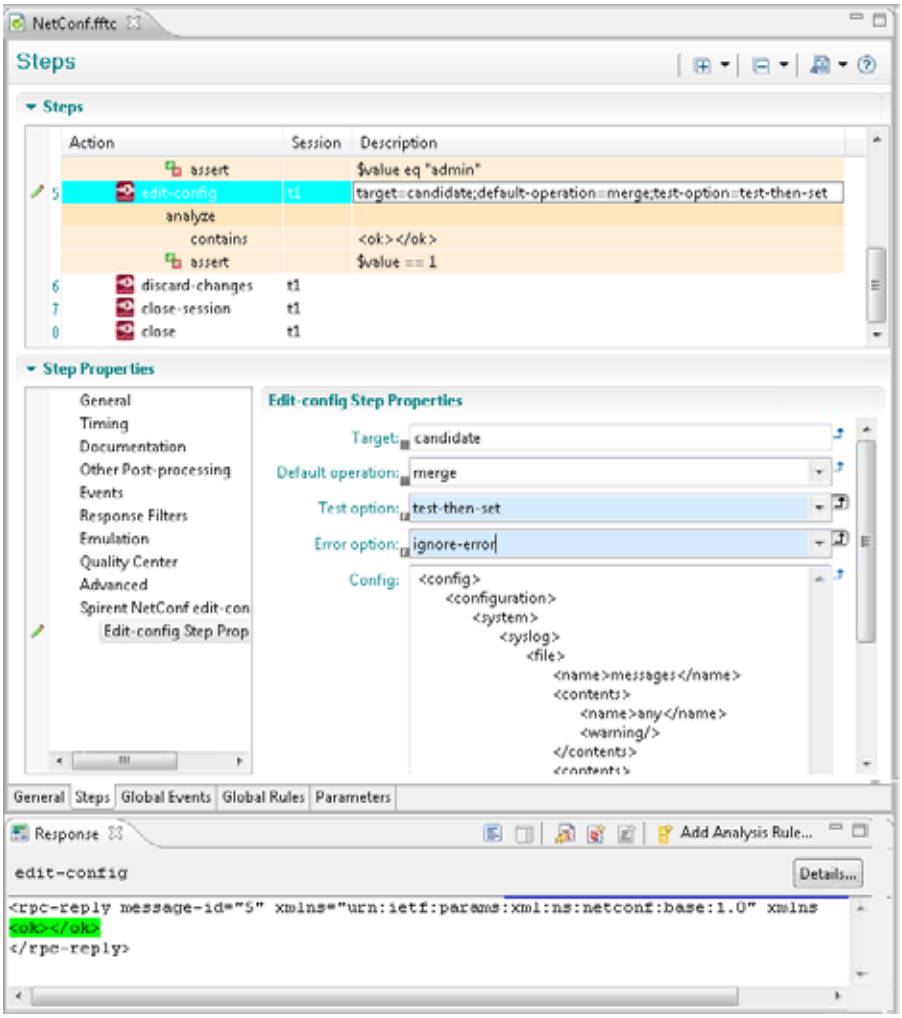
CHAPTER 55

NetConf Sessions

NetConf session window

The NetConf session window displays your commands and the device's responses. You can think of the session window as a terminal — a terminal to a NetConf service as a subsystem that iTest is monitoring and capturing.

Example NetConf step (with Analysis Rule)



NetConf commands

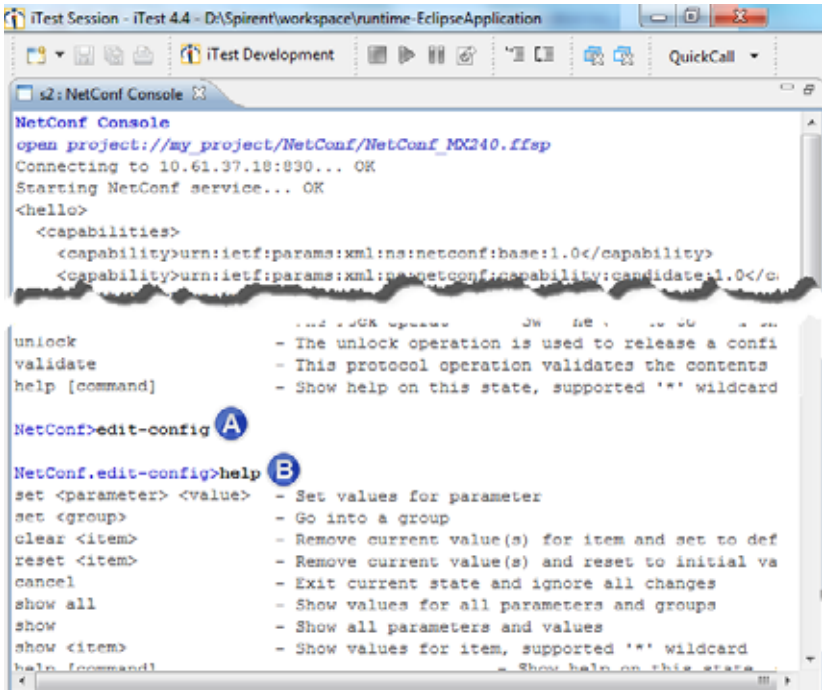
The supported NetConf commands are based on RFC 6241:

https://tools.ietf.org/html/rfc6241

If you use an implementation that does not follow RFC6241, use the iTest **command** action and provide the NetConf command details as an RPC (remote procedure call) argument.

Command reference

- To view the list of supported commands and their descriptions, enter **help** at the command line.
- To view arguments for a command:
  - **a** Enter the command name (this enters the command state — **edit-config** in our example)
  - **b** Enter **help**



Tips for interactive sessions

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper

state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Session profile property settings for NetConf sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Spirent NetConf

IP address	Required. Specify the IP address or hostname for the session with the remote host.
Port	Required. Specify the port for the session (number between 1 and 65535). Default: 22
User	Required. Specify the username used to connect to the remote host.
NetConf service	Specify the name of the NetConf service to start as subsystem. For example: Cisco Nexus 5000 uses **xmlagent** Juniper MX240 uses **netconf**
SSH version	Required. Specify the SSH version. You must further specify authentication settings on the **SSH authentication** property pages. Default: Auto Auto: When iTest connects to the SSH server, they negotiate to determine the SSH version that they both support SSHv1: SSH Version 1 (not recommended) SSHv2: SSH Version 2
NetConf version	Required. Default: 1.0 Select the NetConf version with which you required to work. iTest provides you with two options of NetConf version for your use. Options: 1.0 and 1.1
SSH authentication	Required. Specify the type of authentication that the server allows. When you specify an authentication type, you must specify further authentication settings, as described in the following tables. **Password** (SSHv1 or SSHv2) (default) **KeyboardInteractive** (SSHv2) **PublicKey** (SSHv2) **HostBased** (SSHv1 or SSHv2) **ChallengeResponse** (SSHv1)

SSH authentication: Password

Password	Specify the password used to connect to the remote host. By default, the text is encrypted (masked) here and in all locations where it is used.
Use credentials file	If you configure the **Password** authentication type, then you have the option to configure iTest to go to a specified text file to obtain the latest correct credentials. **Note** The authentication file is not encrypted. Check the box to use the credentials that appear in the file that you specify in the **Credentials file** property. When the box is checked, the settings of the **User** and **Password** properties are ignored.
Credentials file	Specify the path and filename of the text file that holds the credentials to use to log in. **<username>**[space]**<password>** **Note** In the text file, if there is any blank or extra space before the credentials, iTest displays an error after starting the session. See the **Use credentials file** property.

SSH authentication: Keyboard Interactive

Answers	Specify the answers to use when responding to the server challenge. To specify multiple answers, click **Details** and type one answer per line. The text is masked here and in all locations in which it is used.

SSH authentication: PublicKey

Key file	Specify the path to the client key file. Specify a URI that starts with **file://** or **project://**
Passphrase	Optional. Specify the passphrase to use to access the private key in the key file. The text is masked here and in all locations in which it is used.

SSH authentication: HostBased

Key File	Specify the path to the client key file. Specify a URI that starts with **file://** or **project://**
Passphrase	Optional. Specify the passphrase to use to access the private key in the key file. The text is masked here and in all locations in which it is used.

Client hostname	Required for SSHv2 Optional for SSHv1 Specify the hostname of the client
Client username	Specify the user name on the client's computer. For example, you have logged in to the client as username **MyName** and want to connect to the remote host with username **Spirent**. Specify **Spirent** as the **User** property value and **MyName** for **Client username**.

SSH authentication: ChallengeResponse

Challenge answer	Specify the answer to provide when responding to the server challenge. The text is masked here and in all locations in which it is used.

SSH > Connect

Connect timeout	Specify how long to wait (in seconds) for the session to start. Default: 30 seconds
Retry count	Specify how often to retry the connection when the connection attempt times out. Default: 1
Seconds between keepalives	Some devices are configured to close a session if no traffic occurs for a specified period. To ensure that the session is not auto-closed, iTest can send keepalive signals during periods of silence on the line. Specify the number of seconds that should elapse between keepalives. Default: 0 (do not send keepalives)

SSH > Local Port Forwarding

Enable local port forwarding	Check the box to enable local port forwarding. If you enable local port forwarding, then you must specify port forwarding pair information in the **Port forwarding list** text box. Default: unchecked
Port forwarding list	If you enable local port forwarding, then for each port forwarding pair, provide host and port information in the following format (one pair per line): **[localIPaddress:]localPort:remoteIPaddress_or_hostName:remotePort** Notice that **localIPaddress:** is optional. Field substitutions are supported in any part of the text.

SSH > Remote Port Forwarding

Enable remote port forwarding	Check the box to enable remote port forwarding. If you enable remote port forwarding, then you must specify port forwarding pair information in the **Port forwarding list** text box.
Port forwarding list	If you enable remote port forwarding, then for each port forwarding pair, provide host and port information in the following format (one pair per line): **[remoteIPaddress:]remotePort:localIPaddress_or_hostName:localPort** Notice that **remoteIPaddress:** is optional. Field substitutions are supported in any part of the text. Default: unchecked

SSH > X11 Forwarding

Enable X forwarding	Check the box to enable X forwarding. If you enable X forwarding, then you must identify the display in the **X display location** text box. Default: unchecked
X display location	If you enable X forwarding, then type the information that identifies the display in the following format: **[hostName]:displayNumber** Notice that **hostName** is optional. Field substitutions are supported in any part of the text. If you do not specify a value, then iTest uses **127.0.0.1:0** Default: [blank]

CHAPTER 73

Spirent SmartBits sessions

Spirent SmartBits session window

The SmartBits session window is an interactive terminal where you enter commands to perform SmartBits actions on the device. SmartBits returns text responses. (See “Example SmartBits session” on page 1342.)

Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent SmartBits traffic generator devices.

iTest does not use the SmartBits protocol to collect or analyze the traffic data—other steps in your test case can do that.

Tip iTest auto-maps SmartBits device responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries.

---

iTest supports multiple SmartBits sessions within a single instance of iTest. Sessions must not use the same cards/ports.

SmartBits software limitation

SmartBits is not supported on Linux.

Interactive SmartBits sessions

- To view the list of commands while working in a session, type **help** at the prompt.
- View the “Spirent SmartBits command set” on page 1335
- View an “Example SmartBits session” on page 1342
- To view detailed help about syntax and arguments, type the command string followed by **?**

  For example, to view help on the **stream send** command, type **stream send ?**
- Slot and port numbering starts with zero (zero-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.
- Use the * wildcard character to represent all slots or ports

Creating SmartBits test case steps

1. Configure the traffic generator device in the usual way, using its native interface.

2 Save the SAI file: Click **File > Export**, and then save the file. Remember the directory, as you will use it in the next step.

3 Create a SmartBits step. Set the **Action** to **open**. Set the **Command** to the SmartBits session type URI (**application://com.fnfr.svt.applications.smartbits**).

4 As needed, create additional SmartBits steps. When you specify a particular action in the **Action** cell, the **Command** cell may provide hints for the required command text.

As in the Spirent software, slot and port numbering starts with zero (zero-based). Use **slot:port,slot:port,slot:port** syntax to specify slots and ports. Use the * wildcard character to represent all slots or ports.

Spirent SmartBits command set

This topic describes all SmartBits commands that you can submit from iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze the values of interest in the response. See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.

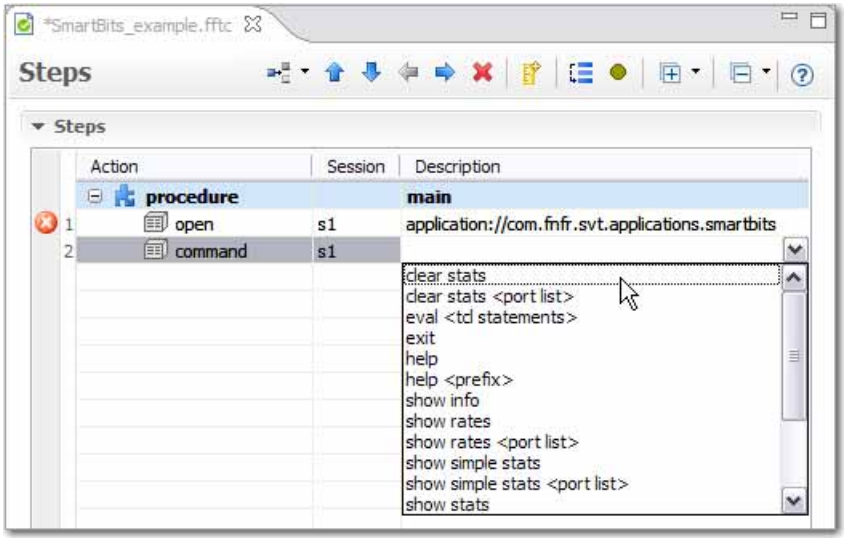
While working in an interactive session

- To view the list of commands while working in an interactive SmartBits session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **show stats** command, type **show stats ?**.

While working on a test case in the Test Case editor

1 For a step in an SmartBits session, in the **Action** cell, select **command**.

2 Select the command from the drop-down list in the **Description** cell.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Command reference

You can use the following commands when defining steps in a test case.

- To view the list of commands while working in an interactive SmartBits session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **show stats** command, type **show stats ?**.
- Slot and port numbering starts with zero (zero-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.

Use the * wildcard character to represent all slots or ports

clear stats [*portList*]	Clear port statistics on all or specified ports
eval *TclStatements*	Evaluate Tcl statements using the device's interpreter
exit	Exit the application and close the session
help [*commandPrefix*]	Display SmartBits session commands
show info	Display device information
show rates [*portList*]	Display port rate statistics for all or specified ports. The **show rates** command does not return L3 counters.
show simple stats [*portList*]	Display a summary of port statistics for all or specified ports
show stats [*portList*]	Display port statistics for all or specified ports
show status	Display status of device slots
transmit start [*portList*]	Start generating traffic on all ports for all or specified ports
transmit stop [*portList*]	Stop generating traffic on all ports for all or specified ports

Creating a Spirent SAI configuration file

To enable a iTest test case to configure the SmartBits device during execution, you edit the session profile to specify which **SAI configuration file** to use.

You can specify the SAI file in SmartBits session profiles or the **Spirent Open Step** properties in the Test Case editor. As a result, when you start a session manually or the test case executes, the **open** step configures the device exactly as if you had configured it using SmartBits.

Note iTest uses only the configuration portion of the SAI file and does not send any test steps from the SAI file to the device.

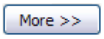
- SAI files cannot contain multiple IP addresses.

1. Configure the Spirent device in the usual way, using its native SmartBits or SmartFlow interface.
2. Now, save the SAI file: In SmartBits, click **File > Export** and then save the file. Remember the directory path, as you will use it in the next step.
3. In the session profile, specify the path in the **SAI location** field.

Session profile property settings for Spirent SmartBits sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Preparing the SAI file

When a session starts, it configures the Spirent device by submitting an SAI file.

- See “Creating a Spirent SAI configuration file” on page 1336 for instructions on generating the SAI file.
- You must import the SAI file into your workspace so that you can specify its URI while defining the session profile.

SmartBits

Chassis IP	Specify the IP address or hostname of the Spirent device.
Initialization script	Specify the path to an initialization script for sessions that use this profile.
SAI location	Specify the path to an SAI configuration file for sessions that use this profile. As a result, the **open** step configures the device exactly as if you had configured it using SmartBits. **Note**: SAI files that contain multiple IP addresses are not supported. See “Creating a Spirent SAI configuration file” on page 1336.
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Tcl Interpreter

Sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the session profile, or you can set a preference for all sessions.

For example, if SmartBits software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the SmartBits session. If, instead, you change the preference for all SmartBits sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Spirent sessions, use the Preference page.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Example SmartBits session

```
Spirent SmartBits command interpreter.
Copyright (c) 2005 - 2012, Spirent Communications, Inc.

Working platform: Windows

Checking smartlib library location in PATH... OK
     'smartlib' library located at 'C:\Program Files\SmartBits\SmartBits API\
Bin\smartlib.tcl'

Connecting to chassis at 10.155.2.246... OK

Reserving Slot 0... OK
Reserving Slot 1... OK

smartbits>help
     clear stats             - Clear port statistics on all ports
     clear stats <port list>     - Clear port statistics
    eval <tcl statements>     - Evaluate Tcl statements using device interpreter
     exit                - Exit the application and close the session
     help                - Display application commands
     help <prefix>           - Display application commands
     show info             - Display device information
     show rates            - Display port rate statistics
     show rates <port list>      - Display port rate statistics
     show stats            - Display port statistics
     show stats <port list>      - Display port statistics
     show status            - Display status of device slots
     transmit start           - Start to generate traffic on all ports
     transmit start <port list>    - Start to generate traffic
     transmit stop           - Stop generating traffic on all ports
     transmit stop <port list>    - Stop generating traffic

smartbits>show stats
Parameter | 0:0:0       | 0:0:1       | 0:1:0      | 0:1:1
------------------------------------------------------------------------------
-----
RcvPkt   | 19265384      | 19388490      | 0        | 0
TmtPkt   | 19388490      | 19265384      | 1922481     | 1922004
Collision | 0          | 0          | 0        | 0
RcvTrig  | 19265384      | 19388490      | 0        | 0
RcvByte  | 1232984576     | 1240863360     | 0        | 0
CRC     | 0          | 0          | 0        | 0
Align    | 0          | 0          | 0        | 0
Oversize  | 0          | 0          | 0        | 0
Undersize | 0          | 0          | 0        | 0


smartbits>clear stats
Clearing statistics on port 0:0:0... OK
Clearing statistics on port 0:0:1... OK
```

```
Clearing statistics on port 0:1:0... OK
Clearing statistics on port 0:1:1... OK

smartbits>show stats
Parameter | 0:0:0       | 0:0:1       | 0:1:0      | 0:1:1
-------------------------------------------------------------------------------
-----
RcvPkt   | 0          | 0          | 0         | 0
TmtPkt   | 0          | 0          | 0         | 0
Collision | 0          | 0          | 0         | 0
RcvTrig  | 0          | 0          | 0         | 0
RcvByte  | 0          | 0          | 0         | 0
CRC     | 0          | 0          | 0         | 0
Align    | 0          | 0          | 0         | 0
Oversize  | 0          | 0          | 0         | 0
Undersize | 0          | 0          | 0         | 0


smartbits>transmit start
Starting transmit on port 0:0:0... OK
Starting transmit on port 0:0:1... OK
Starting transmit on port 0:1:0... OK
Starting transmit on port 0:1:1... OK

smartbits>show stats
Parameter | 0:0:0       | 0:0:1       | 0:1:0      | 0:1:1
-------------------------------------------------------------------------------
-----
RcvPkt   | 7292519      | 12248537     | 0         | 0
TmtPkt   | 8542706      | 10998364     | 1413860     | 1635611
Collision | 0          | 0          | 0         | 0
RcvTrig  | 7292522      | 12248540     | 0         | 0
RcvByte  | 466721344     | 783906496     | 0         | 0
CRC     | 0          | 0          | 0         | 0
Align    | 0          | 0          | 0         | 0
Oversize  | 0          | 0          | 0         | 0
Undersize | 0          | 0          | 0         | 0


smartbits>show stats
Parameter | 0:0:0       | 0:0:1       | 0:1:0      | 0:1:1
-------------------------------------------------------------------------------
-----
RcvPkt   | 24616251     | 28857794     | 0         | 0
TmtPkt   | 25866357     | 27607700     | 2967621     | 3171511
Collision | 0          | 0          | 0         | 0
RcvTrig  | 24616255     | 28857797     | 0         | 0
RcvByte  | 1575440256    | 1846898944    | 0         | 0
CRC     | 0          | 0          | 0         | 0
Align    | 0          | 0          | 0         | 0
Oversize  | 0          | 0          | 0         | 0
Undersize | 0          | 0          | 0         | 0


smartbits>transmit stpop
Invalid arguments. Type "transmit ?" for a list of subcommands.

smartbits>transmit stop
Stopping transmit on port 0:0:0... OK
Stopping transmit on port 0:0:1... OK
Stopping transmit on port 0:1:0... OK
```

```
Stopping transmit on port 0:1:1... OK

smartbits>show stats
Parameter | 0:0:0       | 0:0:1       | 0:1:0     | 0:1:1
-------------------------------------------------------------------------------
-----
RcvPkt   | 47718846      | 48261427      | 0        | 0
TmtPkt   | 48261427      | 47718846      | 4765450    | 4758749
Collision | 0          | 0          | 0        | 0
RcvTrig  | 47718846      | 48261427      | 0        | 0
RcvByte  | 3054006144    | 3088731328    | 0        | 0
CRC     | 0          | 0          | 0        | 0
Align    | 0          | 0          | 0        | 0
Oversize  | 0          | 0          | 0        | 0
Undersize | 0          | 0          | 0        | 0


smartbits>show info
Device information:
     Model: SMB600
     S/N: 06000798
     Firmware: 2.80.0003

Device ports:
     0:0:0, 0:0:1, 0:1:0, 0:1:1

Session ports:
     0:0:0, 0:0:1, 0:1:0, 0:1:1

smartbits>show status
Slot 0 is reserved by current user
Slot 1 is reserved by current user

smartbits>
```

Setting preferences for Spirent SmartBits sessions

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Spirent > Session Types > Spirent SmartBits**.

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Spirent SmartBits

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if SmartBits software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the SmartBits session. By changing the preference, you can avoid changing the properties in the session profile, but your session

profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	Default: **Auto-select** **Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter.
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked
Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Default: unchecked Use remote shell logging.

CHAPTER 74

Spirent TestCenter CLI sessions (Obsolete and Deprecated)

Important

The Spirent TestCenter CLI session type is ***obsolete and is no longer under development***.

This chapter is provided only to support existing implementations. You should develop new test cases using the Spirent TestCenter session type (described in Chapter 75, “Spirent TestCenter sessions”).

Spirent TestCenter CLI session window

The TestCenter CLI session window is an interactive terminal where you enter commands to perform TestCenter actions on the device. TestCenter returns text responses.

Because all commands and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent TestCenter traffic generator devices.

iTest does not use the TestCenter protocol to collect or analyze the traffic data—other steps in your test case can do that.

iTest auto-maps TestCenter device responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. See “Analysis rules: Validating responses and setting Pass / Fail” on page 690, “Data view” on page 326, and “Structure view” on page 358.

Spirent TestCenter CLI command set

This topic describes all TestCenter CLI commands that you can submit from iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze the values of interest in the response.

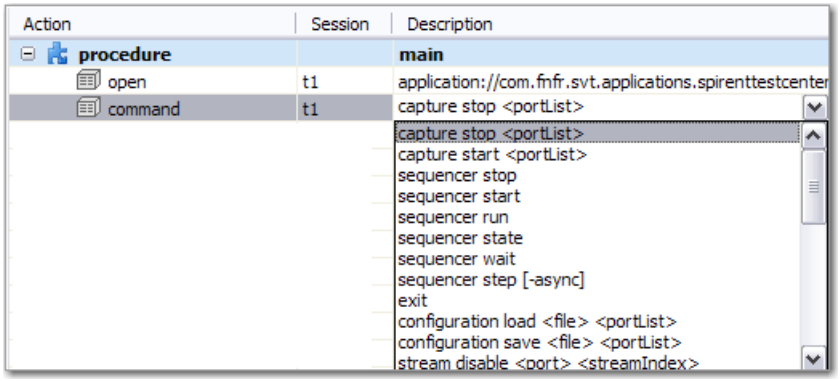
Note To monitor the Rx and Tx ports for a particular stream block, subscribe to Rx Port pair results. Do not confuse this information with the Rx Stream Block result (which is used to specify which Stream block belongs to which Port). For more details, refer to Spirent documentation.

Interactive TestCenter CLI commands

- To view the list of commands while working in a session, type **help** at the prompt.
- View the full “Spirent TestCenter CLI command set” on page 1346
- View an “Example TestCenter CLI session” on page 1357
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **stream send** command, type **stream send ?**.
- Slot and port numbering starts with zero (zero-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.
- Use the * wildcard character to represent all slots or ports. For example, to specify all ports on card **2**, use **2:***

While working on a test case in the Test Case editor

1. For a step in a TestCenter CLI session, in the **Action** cell, select **command**.
2. Select the command from the drop-down list in the **Description** cell.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Command reference

You can use the following commands when defining steps in a test case.

- Slot and port numbering starts with 1 (1-based).
- Use **slot:port,slot:port,slot:port** syntax to specify slots and ports.

Use the * wildcard character to represent all slots or ports. For example, to specify all ports on card **2**, use **2:***

capture save *port filename*	Saves capture data from the specified port to the specified file. **Note** Spirent limits the path to 256 characters.
capture start *[portList]*	Starts capturing on all or specified ports
capture stop *[portList]*	Stops capture on all or specified ports.
clear stats	Clears statistics for all ports
configuration load *filePath*	Configures the device with information from the specified file. **Note** Spirent limits the path to 256 characters. The **configuration load** command supports XML configuration files generated by **configuration save**. If the filename extension is **.xml**, then iTest interprets the configuration file as XML format. iTest expects that a second file exists in the same directory. The second file must have the same name postfixed with **_logic**. For example, **config.cfg** and **config.cfg_logic** During interactive TestCenter sessions, **configuration load** performs replacements on the file so that the specific information (like IP address) are translated into variables or command substitutions. Arguments Name Description Example filePath Configuration file to load project:///spirent.xml

configuration save *filePath*	Save the current configuration information returned by the device into a specified file. **Note** Spirent limits the path to 256 characters. The file is used to configure the device exactly as if you had configured it using TestCenter: In test cases, you use **configuration load** to load the configuration settings to the device. You can specify the resulting file in the **Configuration file** setting in a session profile. The configuration is performed when the session starts. The configuration file format depends on the filename extension of the specified file. If the extension is **.xml**, then the configuration is saved in XML format. If the target format is XML, you can provide the list of ports to include in the configuration file. Arguments Name Description Example filePath Configuration file to save the config to project:///spirent.xml
eval *Tcl_statement*	Evaluate a Tcl statement for direct access to Spirent TestCenter's TCL API
sequencer start	Gets the sequencer and issues the commands to chassis to start the sequencer. Immediately returns control to the user.
sequencer stop	Stops the current sequence execution.
sequencer step [-async]	Executes the next step in the sequence. If the **-async** option is provided, the step is started and the control is returned to the user immediately. In this case, use sequencer wait to wait for the step to finish.
sequencer wait	Waits for the current step in the named sequence to finish. Blocks the call until step is completed.
sequencer state	Returns the current state of the named sequence. Possible values: IDLE, WAIT, and PAUSE
sequencer run	Issues command to run all the steps to end. This command blocks until the sequence run is finished. It prints out a message for each step in the sequencer as the steps execute. If you press Ctrl-C, the sequencer run is aborted.
show info host	Shows information about the 'host' parent object
show info router	Shows information about the 'router' parent object. (Replaces **get routerinfo**)
show info stream-block *[portList]*	Shows streams from all or specified ports
show info *trafficGroup*	Shows information about traffic group
show packet *port packetIndex* [*-raw*]	Displays the specified captured packet on the specified port
show results *[objects]*	Shows results of specified type for all or specified objects

show stats *[portList]*	Displays a table of statistics for all ports or by port Shows subscribed stats for the 'port' parent object for the specified port list
source	Sources TCL script from workspace
stream enable *port streamIndex*	Enables transmission on a specified stream
stream disable *port streamIndex*	Disables transmission on a specified stream
subscribe	Subscribes for specific result type or lists all the result-types available
transmit start *[portList]*	Starts transmitting on all or specified ports
transmit stop *[portList]*	Stops transmitting on all or specified ports
unsubscribe	Unsubscribes for specific result type or lists all the result-types available
unsubscribe all	Unsubscribes for all available result types

Spirent TestCenter result types in iTest

This table correlates TestCenter data items with the responses to TestCenter commands in iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or apply analysis rules to the values of interest in the response

UI grouping/location		**iTest result type**
Port Traffic	**Basic Traffic Results**	AnalyzerPortResults, GeneratorPortResults
	Diffserv Results	DiffServResults
	Port Average Latency Results	PortAvgLatencyResults
	Overflow Results	OverflowResults
	Port Pair Results	Tx/RxPortPairResults
	CPU Port Results	Tx/RxCpuPortResults
Stream Results	**Traffic Group Results**	Tx/RxTrafficGroupResults
	Filtered Stream Results	FilteredStreamResults
	Detailed Stream Results	TxStreamResults, RxStreamSummaryResults
	Stream Block Results	Tx/RxStreamBlockResults

Port Protocols	**SONET Interface Results**		SonetResults
	POS Interface Results		PppProtocolResults
	ARPND Results		ArpNdResults
	LACP Results		LacpPortResults
	L2TP Results Not supported in iTest 3.1		L2tpPortResults
	PPPoX Results		PPPoEPortResults
	DHCP Results		Dhcpv4PortResults
	DHCPv6 Results		Dhcpv6PortResults
	IGMP Results		IgmpPortResults
	MLD Results		MldPortResults
	EOAM Results		EoamPortResults
Router Protocols	**BGP Results**		BgpRouterResults
	OSPFv2 Results		Ospfv2Results
	OSPFv3 Results		Ospfv3Results
	ISIS Results		IsisRouterResults
	RIP Results		RipRouterResults
	LDP Results		LdpRouterResults
	RSVP Results		RsvpRouterResults
	PIM Results		PimRouterResults
	IGMP Querier Results		IgmpRouterResults
	MLD Querier Results		MldRouterResults
	STP Results		BridgePortResults
	MSTI Results		BridgePortResults
	LDP-RSVP LSP Results	**LDP LSP Results**	LdpLspResults
		RSVP LSP Results	RsvpLspResults
	EOAM Results	**Port Results** Not supported in iTest 3.1	EoamPortResults
		MEG Results Not supported in iTest 3.1	EoamMegResults
		CC Results Not supported in iTest 3.1	EoamContChkLocalResults
		LB Results Not supported in iTest 3.1	EoamLoopbackResults
		LT Results Not supported in iTest 3.1	EoamLinkTraceResults

Host Protocols	**DHCP Results**	**Dhcpv4BlockResults**	
	PPPoX Results	**PPP/PPPoeClientBlockResults, PPP/PPPoeServerBlockResults** Not supported in iTest 3.1	
	DHCPv6PD Results	**Dhcpv6BlockResults**	
	L2TP Results	**L2TPv2BlockResults** Not supported in iTest 3.1	
	IGMP Results	**IgmpHostResults**	
	MLD Results	**MldHostResults**	
	IGMP-MLD Group Results	**IGMP Host-Group Results**	IgmpGroupMembershipResults
		MLD Host-Group Results	MldGroupMembershipResults
	SIP Results Not supported in iTest 3.1		SipUaBlockResults
IPTV	**Test Results**		IptvTestResults
	Port Results		IptvPortResults
	STB Block Results		IptvStbBlockResults
	Viewing Profile Results		IptvViewingProfileResults
	Channel Results		IptvChannelResults

Session profile property settings for Spirent TestCenter CLI sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the Session Profile editor). To access the other settings, click More >>.

Note If you have already saved a session profile with the appropriate settings, then you do not need to configure another session profile to start a session. Instead, you can quickly start a session in either of the following methods:

- In the Favorites view, double-click the existing session profile
- In the iTest Explorer, right-click the existing session profile and select **Start**

Spirent TestCenter

Spirent chassis IP address	Specify the IPaddress or DNS hostname of the device.
Cards (slots)	When you select a slot, all of its ports are selected by default. You can also select individual ports using the Ports property. **Note**: If you specify **Cards**, then do not specify **Ports**. Slot and port numbering starts with zero (it is zero-based). Use <slot>:<port>,<slot>:<port>,<slot>:<port> syntax to specify slots and ports. Examples To specify card 2, port 1, use 2:1. To specify multiple cards and ports, use 2:1,3:2,4:3
Ports	Specify the port for the session (a single number or series of numbers separated by commas). **Note** If you specify **Ports**, then do not specify **Cards**. **Important** iTest assigns ports in the order listed. For example, "1:9,1:10" assigns Port 9 first and "1:10,1:9" assigns port 10 first. If you are loading a configuration file and port order is important, you must specify ports in the same order as in the configuration file.
Force to take ownership	Upon connecting, take ownership so no other user can submit commands.
Configuration file	Optional: Specify the configuration file to use to configure the device when the session starts. You generate a configuration file using the TestCenter **configuration save** command.

Tcl Interpreter

Sessions rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used. You can specify the path to the correct interpreter here, in the session profile, or you can set a preference for all sessions.

For example, if SmartBits software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter here, in the session profile used to start the SmartBits session. If, instead, you change the preference for all SmartBits sessions, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

To configure these settings for all Spirent sessions, use the Preference page.

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**

Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Example TestCenter CLI session

```
Spirent TestCenter CLI command interpreter. Copyright (c) 2005 - 2013, Spirent


Using external tcl interpreter
tcl version: 8.4.5
tclsh location: C:/Program Files/Tcl845/bin/tclsh84.exe
tcl library: C:/Program Files/Tcl845/lib/tcl8.4
tcl package search path: {C:/Program Files/Ixia/TclScripts/lib} {C:/Program
Files/Tcl83/lib} {C:/Program Files/Spirent Communications/Spirent TestCenter
```

```
2.01/Spirent TestCenter Application/} {C:/Program Files/Tcl845/lib/tcl8.4}
{C:/Program Files/Tcl845/l
ib}


Loading TCL package 'SpirentTestCenter'...
    2.0
Loading TCL package 'SpirentTestCenter' finished with OK

Loading TCL helper scripts...OK

Connecting to chassis at 10.155.2.245...OK

Getting chassis info...
Getting chassis info finished with OK

Loading configuration file D:\work\itest\project\routers_seq.xml...
    Preparing XML config file... OK
    Disconnecting from chassis... OK
    Loading project... OK
    Attaching to physical ports... OK
    Applying configuration... OK
    Refreshing session ports... OK
Loading configuration file D:\work\itest\project\routers_seq.xml finished
with OK

Card(1) 800-5135-5:N05139639:2.01.3037:MODULE_STATUS_UP:10/100/1000 DUAL
MEDIA, 4 PORT
        PhysicalPortGroup 3 OWNERSHIP_STATE_AVAILABLE MODULE_STATUS_DOWN
Reserved:false default@cldvmxp01
                PhysicalPort 5 //10.155.2.245/1/1
                PhysicalPort 6 //10.155.2.245/1/2
        PhysicalPortGroup 4 OWNERSHIP_STATE_RESERVED MODULE_STATUS_UP
Reserved:true komaz@komazz
                PhysicalPort 7 //10.155.2.245/1/3
                PhysicalPort 8 //10.155.2.245/1/4
Card(2) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(3) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(4) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(5) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(6) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(7) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(8) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(9) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(10) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(11) ::0:MODULE_STATUS_UNKNOWN:<slot empty>
Card(12) ::0:MODULE_STATUS_UNKNOWN:<slot empty>


Spirent TestCenter>help
capture start <portList>               - Starts capture on all or specified
ports
capture stop <portList>                - Stops capture on all or specified ports
clear stats <portList>                 - Clears statistics on all or specified
ports
configuration load <file> <portList>   - Loads configuration from TCL or XML
file
configuration save <file> <portList>   - Saves current configuration to TCL or
XML
eval <Tcl expression>                  - Evaluates TCL expression
exit                                    - Closes session
```

```
help <prefix>                            - Displays help information
sequencer run                            - Issues command to run all the steps to
end. This command blocks till the sequence run is finished.
sequencer start                          - Starts sequencer
sequencer state                          - Returns state of sequencer
sequencer step [-Specify how long to wait for the prompt to appear
]                - Executes one step from sequencer
sequencer stop                           - Stops sequencer
sequencer wait                           - Waits for the current step in the named
sequence to finish. Blocks the call till step is completed.
show info host <portList>                - Shows information about the host
show info router <portList>              - Shows information about the router
show info stream-block <portList>        - Shows streams on all or specified ports
show info traffic-group                  - Shows traffic groups
show results <resultType> <parent>       - Displays results of specified type from
all or specified objects
show stats port <portList>               - Shows statistics from all or specified
ports
show stats stream-block <portList>       - Shows statistics about the specified
stream block
source <file>                            - Sources TCL script
stream disable <port> <streamIndex>      - Disables stream
stream enable <port> <streamIndex>       - Enables stream
subscribe <resultType>                   - Subscribes for provided result type
transmit start <portList>                - Starts transmit on all or specified
ports
transmit stop <portList>                 - Stops transmit on all or specified
ports
unsubscribe <resultType>                 - Unsubscribes from provided result
type
unsubscribe all                          - Unsubscribes from all result types

Spirent TestCenter>show info host
EthIIIf
Port                |                  1:3               |1:4
--------------------+-------------------------------------+----------------
--
Host                |Host 11           Host 3            |Host 4
--------------------+-------------------------------------+----------------
--
Active              |true              true              |true
AuthenticationState |RESOLVE_DONE      RESOLVE_DONE      |RESOLVE_DONE
Authenticator       |default           default           |default
EffectiveBlockCount |1                 1                 |1
IfCountPerLowerIf   |1                 1                 |1
IfState             |READY             READY             |READY
IsLoopbackIf        |false             false             |false
IsRange             |true              true              |true
Name                |EthIIIf 11        EthIIIf 3         |EthIIIf 4
SourceMac           |00:10:94:00:00:01 00:10:94:00:00:02 |00:10:94:00:00:03
SrcMacList          |                                    |
SrcMacRepeatCount   |0                 0                 |0
SrcMacStep          |00:00:00:00:00:01 00:00:00:00:00:01 |00:00:00:00:00:01
SrcMacStepMask      |00:00:ff:ff:ff:ff 00:00:ff:ff:ff:ff |00:00:ff:ff:ff:ff

Ipv4If
Port                       |1:3              1:4
---------------------------+-----------------------------------
Host                       |Host 3           Host 4
---------------------------+-----------------------------------
Active                     |true             true
```

```
AddrList                  |
AddrRepeatCount           |0                  0
AddrResolveState          |RESOLVE_DONE       RESOLVE_DONE
AddrResolver              |default            default
AddrStep                  |0.0.0.1            0.0.0.1
AddrStepMask              |0.0.0.255          0.0.0.255
Address                   |192.85.1.3         192.85.2.3
EffectiveBlockCount       |1                  1
Gateway                   |192.85.1.1         192.85.2.1
GatewayMac                |00:00:01:00:00:01  00:00:01:00:00:01
GatewayMacResolveState    |RESOLVE_NEEDED     RESOLVE_NEEDED
GatewayMacResolver        |default            default
IfCountPerLowerIf         |1                  1
IfState                   |READY              READY
IsLoopbackIf              |false              false
IsRange                   |true               true
Name                      |Ipv4If 3           Ipv4If 4
NeedsAuthentication       |false              false
PrefixLength              |24                 24
ResolveGatewayMac         |true               true
SkipReserved              |true               true
Tos                       |192                192
Ttl                       |10                 10
UsePortDefaultIpv4Gateway |false              false

Ipv6If
Port                      |1:3
--------------------------+----------------------
Host                      |Host 11
--------------------------+----------------------
Active                    |true
AddrList                  |
AddrRepeatCount           |0
AddrResolveState          |RESOLVE_DONE
AddrResolver              |default
AddrStep                  |::1
AddrStepMask              |::ffff:ffff:ffff:ffff
Address                   |2000::2
EffectiveBlockCount       |1
FlowLabel                 |7
Gateway                   |2000::1
GatewayMac                |00:00:01:00:00:01
GatewayMacResolveState    |RESOLVE_NEEDED
GatewayMacResolver        |default
HopLimit                  |10
IfCountPerLowerIf         |1
IfState                   |READY
IsLoopbackIf              |false
IsRange                   |true
Name                      |Ipv6If 11
NeedsAuthentication       |false
PrefixLength              |64
ResolveGatewayMac         |true
TrafficClass              |0
UsePortDefaultIpv6Gateway |false

Spirent TestCenter>show info router
BgpRouterConfig
Port                |                 1:3              |
1:4              |
--------------------+-----------------------------------+---------------
```

```
---------------------+
Router                |Router 1          Router 2          |Router 3
Router 4          |
---------------------+-----------------------------------+--------------
---------------------+
Active                |true              true              |true
true              |
Afi                   |0                 0                 |0
0                 |
AsNum                 |1                 1                 |1
1                 |
DutIpv4Addr           |192.85.1.1        192.85.1.1        |192.85.1.1
192.85.1.1        |
DutIpv6Addr           |::2               ::2               |::2
::2               |
EiBgp                 |EBGP              EBGP              |EBGP
EBGP              |
GracefulRestart       |false             false             |false
false             |
HoldTimeInterval      |90                 90                |90
               90                |
Initiate              |true              true              |true
true              |
IpVersion             |IPV6              IPV6              |IPV6
IPV6              |
KeepAliveInterval     |30                30                |30
30                |
MinLabel              |16                16                |16
16                |
Name                  |BgpRouterConfig 1 BgpRouterConfig 2 |BgpRouterConfig 3
BgpRouterConfig 4 |
PeerAs                |1001              1001              |1001
1001              |
RestartTime           |90                90                |90
90                |
RouterState           |NONE              NONE              |NONE
NONE              |
SubAfi                |0                 0                 |0
0                 |
UsePartialBlockState|false               false             |false
false             |
ViewRoutes            |false             false             |false
false             |

EthIIIf
Port                  |                  1:3               |
1:4               |
--------------------+-----------------------------------+---------------
--------------------+
Router                |Router 1          Router 2          |Router 3
Router 4          |
--------------------+-----------------------------------+---------------
--------------------+
Active                |true              true              |true
true              |
AuthenticationState |RESOLVE_DONE      RESOLVE_DONE      |RESOLVE_DONE
RESOLVE_DONE      |
Authenticator         |default           default           |default
default           |
EffectiveBlockCount |1                 1                 |1
1                 |
```

```
IfCountPerLowerIf   |1                1                |1
1                |
IfState             |READY            READY            |READY
   READY            |
IsLoopbackIf        |false            false            |false
false            |
IsRange             |true             true             |true
true             |
Name                |EthIIIf 6        EthIIIf 7        |EthIIIf 8
EthIIIf 9        |
SourceMac           |00:10:94:00:00:01 00:10:94:00:00:02 |00:10:94:00:00:03
00:10:94:00:00:04 |
SrcMacList          |
|                                    |
SrcMacRepeatCount   |0                 0                |0
0                |
SrcMacStep          |00:00:00:00:00:01 00:00:00:00:00:01 |00:00:00:00:00:01
00:00:00:00:00:01 |
SrcMacStepMask      |00:00:ff:ff:ff:ff 00:00:ff:ff:ff:ff |00:00:ff:ff:ff:ff
00:00:ff:ff:ff:ff |

Ipv6If
Port                    |                    1:3
|                     1:4                  |
-------------------------+-----------------------------------------+--
----------------------------------------+
Router                   |Router 1             Router 2
|Router 3             Router 4             |
-------------------------+-----------------------------------------+--
----------------------------------------+
Active                   |true                 true
|true                 true                 |
AddrList                 |
|                                     |
AddrRepeatCount          |0                    0
|0                  0                    |
AddrResolveState         |RESOLVE_DONE         RESOLVE_DONE
|RESOLVE_DONE         RESOLVE_DONE         |
AddrResolver             |default              default
|default              default              |
AddrStep                 |::1                  ::1
|::1                  ::1                  |
AddrStepMask             |::ffff:ffff:ffff:ffff ::ffff:ffff:ffff:ffff
|::ffff:ffff:ffff:ffff ::ffff:ffff:ffff:ffff |
Address                  |2000::2              2000::3
|2000:0:1::2          2000:0:1::3          |
EffectiveBlockCount      |1                    1
|1                    1                    |
FlowLabel                |7                    7
|7                    7                    |
Gateway                  |2000::1              2000::1
|2000:0:1::1          2000:0:1::1          |
GatewayMac               |00:00:01:00:00:01    00:00:01:00:00:01
|00:00:01:00:00:01    00:00:01:00:00:01    |
GatewayMacResolveState   |RESOLVE_NEEDED       RESOLVE_NEEDED
|RESOLVE_NEEDED       RESOLVE_NEEDED       |
GatewayMacResolver       |default              default
|default              default              |
HopLimit                 |10                   10
|10                   10                   |
IfCountPerLowerIf        |1                    1
```

```
|1                    1                    |
IfState                 |READY                READY
|READY                 READY                |
IsLoopbackIf            |false                false
|false                 false                |
IsRange                 |true                 true
|true                  true                 |
Name                    |Ipv6If 7             Ipv6If 9
|Ipv6If 11             Ipv6If 13            |
NeedsAuthentication     |false                false
|false                 false                |
PrefixLength            |64                   64
|64                    64                   |
ResolveGatewayMac       |true                 true
|true                  true                 |
TrafficClass            |0                    0
|0                     0                    |
UsePortDefaultIpv6Gateway |false              false
|false                 false                |

MplsIf
Port                |
       1:3
|
1:4                                  |
--------------------+-------------------------------------------------
----------------------+-------------------------------------------------
--------------------------+
Router              |Router 1                              Router
2                             |Router 3                              Router
4                             |
--------------------+-------------------------------------------------
----------------------+-------------------------------------------------
--------------------------+
Active              |true
true                                 |true
                    true                                 |
DstMac              |00:00:01:00:00:01
00:00:01:00:00:01
|00:00:01:00:00:01
00:00:01:00:00:01                    |
EffectiveBlockCount |1
1
|1
1                                    |
ExperimentalBits    |0
0                                    |0
                                   0
|
IfCountPerLowerIf   |1
1
|1
1                                    |
IfState             |READY
READY
|READY
READY                                |
IsLoopbackIf        |false
false
```

```
             |false
false                                  |
IsRange             |false
false
|false                                  false
          |
Label               |1
1
|1
1                                      |
LabelList           |
                              |
                                 |
LabelRepeatCount    |0
0
|0                                      0
                           |
LabelResolveState   |RESOLVE_DONE
RESOLVE_DONE
|RESOLVE_DONE
RESOLVE_DONE                           |
LabelResolver       |Bgp
       Bgp
|Bgp
Bgp                                    |
LabelStep           |1
1                                      |1
    1                                      |
Name                |Router 1 (BGP Router  DUT 192.85.1.1) Router 2 (BGP Router
DUT 192.85.1.1) |Router 3 (BGP Router  DUT 192.85.1.1) Router 4 (BGP Router  DUT
192.85.1.1) |
StackBit            |0
                       0
|0
0                                      |
TTL                 |0
0
```

```
|0
0                                    |

Ospfv3RouterConfig
Port                 |                     1:3
    |                     1:4                 |
---------------------+-----------------------------------------+---------
---------------------------------+
Router               |Router 1             Router 2             |Router
3            Router 4             |
---------------------+-----------------------------------------+---------
---------------------------------+
Active               |true                 true
|true                 true                |
Advertise            |ALL_LSAS             ALL_LSAS
|ALL_LSAS             ALL_LSAS            |
AreaId               |0.0.0.0              0.0.0.0
|0.0.0.0              0.0.0.0             |
FloodDelay           |100                  100
|100                  100                 |
HelloInterval        |10                   10
|10                   10                  |
IfCost               |1                    1
|1                    1                   |
IfId                 |0                    0
|0                    0                   |
InstanceId           |0                    0
|0                    0                   |
LsaRefreshInterval   |1800                 1800
|1800                 1800                |
Name                 |Ospfv3RouterConfig 1 Ospfv3RouterConfig 2
|Ospfv3RouterConfig 3 Ospfv3RouterConfig 4 |
NeighborState        |DOWN                 DOWN                |DOWN
              DOWN                |
NetworkType          |NATIVE               NATIVE
|NATIVE               NATIVE              |
Options              |V6BIT|EBIT|RBIT     V6BIT|EBIT|RBIT
|V6BIT|EBIT|RBIT     V6BIT|EBIT|RBIT     |
RetransmitInterval   |5                    5
|5                    5                   |
RouterDeadInterval   |40                   40
|40                   40                  |
RouterPriority       |0                    0
|0                    0                   |
RouterState          |NONE                 NONE
|NONE                 NONE                |
UsePartialBlockState |false                false               |false
           false               |

RipRouterConfig
Port                 |                 1:3              |
1:4              |
---------------------+-----------------------------------+--------------
---------------------+
Router               |Router 1          Router 2          |Router 3
Router 4          |
---------------------+-----------------------------------+--------------
---------------------+
Active               |true              true              |true
true              |
DutIpv4Addr          |10.1.1.2          10.1.1.2          |10.1.1.2
```

```
10.1.1.2          |
DutIpv6Addr          |ff02::9           ff02::9           |ff02::9
ff02::9           |
InterUpdateDelay     |10                10                |10
10                |
MaxRoutePerUpdate    |25                25                |25
25                |
Name                 |RipRouterConfig 1 RipRouterConfig 2 |RipRouterConfig 3
RipRouterConfig 4 |
RipVersion           |NG                NG                |NG
NG                |
RouterState          |NONE              NONE              |NONE
NONE              |
UpdateInterval       |30                30                |30
30                |
UpdateJitter         |0                 0                 |0
0                 |
UpdateType           |MULTICAST         MULTICAST         |MULTICAST
MULTICAST         |
UsePartialBlockState |false             false             |false
false             |
ViewRoutes           |false             false             |false
false             |

Spirent TestCenter>show info stream-block
StreamBlock                       |1:3/1             1:3/2             1:4/1
----------------------------------+-----------------------------------------
--------------
Active                            |true              true              true
BpsLoad                           |100000000.000000  100000000.000000
100000000.000000
BurstSize                         |1                 1                 1
ConstantFillPattern               |0                 0                 0
EnableBidirectionalTraffic        |false             false             false
EnableControlPlane                |false             false             false
EnableFcsErrorInsertion           |false             false             false
EnableStreamOnlyGeneration        |false             false             false
EnableTxPortSendingTrafficToSelf  |false             false             false
EndpointMapping                   |ONE_TO_ONE        ONE_TO_ONE
ONE_TO_ONE
FillType                          |CONSTANT          CONSTANT
CONSTANT
FixedFrameLength                  |128               128               128
FlowCount                         |5                 5                 1
FpsLoad                           |84459.000000      84459.000000
84459.000000
FrameLengthMode                   |FIXED             FIXED             FIXED
IbgInMillisecondsLoad             |1344.000000       1344.000000
1344.000000
IbgInNanosecondsLoad              |1344.000000       1344.000000
1344.000000
IbgLoad                           |1344.000000       1344.000000
1344.000000
InsertSig                         |true              true              true
InterFrameGap                     |12                12                12
IsArpResolved                     |false             false             false
KbpsLoad                          |100000.000000     100000.000000
100000.000000
Load                              |10.000000         10.000000
10.000000
LoadUnit                          |PERCENT_LINE_RATE PERCENT_LINE_RATE
```

```
PERCENT_LINE_RATE
MaxFrameLength                  |256              256              256
MbpsLoad                        |100.000000       100.000000
100.000000
MinFrameLength                  |128              128              128
Name                            |StreamBlock 1    StreamBlock 7
StreamBlock 2
PercentageLoad                  |10.000000        10.000000
10.000000
Priority                        |0                0                0
RunningState                    |STOPPED          STOPPED
STOPPED
StartDelay                      |0                0                0
State                           |READY            READY            READY
Status                          |None             None             None
StepFrameLength                 |1                1                1
StreamBlockIndex                |0                1                0
StreamCount                     |5                5                1
StreamIndex                     |
Targets                         |ipv4:ipv41       ipv4:ipv42
ipv4:ipv43
TrafficPattern                  |BACKBONE         BACKBONE
BACKBONE
dstbinding-Targets              |ipv4if2          ipv4if2
ipv4if1
srcbinding-Targets              |ipv4if1          ipv4if1
ipv4if2

Spirent TestCenter>show info traffic-group
TrafficGroup |SubGroup |StreamBlocks
-------------+---------+--------------------
Tg1          |Sub1     |1:3/1, 1:4/1, 1:3/2
-------------+---------+--------------------
             |Sub2     |1:3/1, 1:3/2
Tg2          |Sub3     |1:4/1
-------------+---------+--------------------

Spirent TestCenter>subscribe AnalyzerPortResults GeneratorPortResults
Subscribed on result type(s) AnalyzerPortResults, GeneratorPortResults
Spirent TestCenter>transmit start
Successfully started transmit on port(s): 1:3,1:4
Spirent TestCenter>transmit stop
Successfully stopped transmit on port(s): 1:3,1:4
Spirent TestCenter>subscribe
Parent        |Type                        |Subscribed
--------------+----------------------------+-----------
              |IgmpGroupMembershipResults |no
              |IgmpHostResults            |no
Host          |IptvStbBlockResults        |no
              |MldGroupMembershipResults  |no
              |MldHostResults             |no
--------------+----------------------------+-----------
              |Dhcpv4BlockResults         |no
Host, Router  |Dhcpv6BlockResults         |no
--------------+----------------------------+-----------
              |AnalyzerPortResults        |yes
              |ArpNdResults               |no
              |Dhcpv4PortResults          |no
              |Dhcpv6PortResults          |no
              |DiffServResults            |no
              |FilteredStreamResults      |no
```

```
              |GeneratorPortResults       |yes
              |IgmpPortResults            |no
              |IptvPortResults            |no
              |LacpPortResults            |no
Port          |MldPortResults             |no
              |OverflowResults            |no
              |PortAvgLatencyResults      |no
              |PppProtocolResults         |no
              |PppoePortResults           |no
              |RxCpuPortResults           |no
              |RxPortPairResults          |no
              |SonetResults               |no
              |TxCpuPortResults           |no
              |TxPortPairResults          |no
--------------+---------------------------+-----------
              |IptvChannelResults         |no
Project       |IptvTestResults            |no
              |IptvViewingProfileResults  |no
--------------+---------------------------+-----------
              |BgpRouterResults           |no
              |BridgePortResults          |no
              |IgmpRouterResults          |no
              |IsisRouterResults          |no
              |LdpLspResults              |no
              |LdpRouterResults           |no
Router        |MldRouterResults           |no
              |Ospfv2RouterResults        |no
              |Ospfv3RouterResults        |no
              |PimRouterResults           |no
              |RipRouterResults           |no
              |RsvpLspResults             |no
              |RsvpRouterResults          |no
--------------+---------------------------+-----------
              |RxStreamBlockResults       |no
              |RxStreamSummaryResults     |no
Stream-block  |TxStreamBlockResults       |no
              |TxStreamResults            |no
--------------+---------------------------+-----------
              |RxTrafficGroupResults      |no
Traffic-group |TxTrafficGroupResults      |no
--------------+---------------------------+-----------

Spirent TestCenter>show stats port
AnalyzerPortResults
Port                   |1:3     1:4
-----------------------+------------------
ComboTriggerCount      |0       0
ComboTriggerRate       |0       0
FcsErrorFrameCount     |0       0
FcsErrorFrameRate      |0       0
HwFrameCount           |0       0
IcmpFrameCount         |0       0
IcmpFrameRate          |0       0
Ipv4ChecksumErrorCount |0       0
Ipv4ChecksumErrorRate  |0       0
Ipv4FrameCount         |214141  252765
Ipv4FrameRate          |0       0
Ipv6FrameCount         |0       0
Ipv6FrameRate          |0       0
Ipv6OverIpv4FrameCount |0       0
Ipv6OverIpv4FrameRate  |0       0
```

```
JumboFrameCount        |0        0
JumboFrameRate         |0        0
MaxFrameLength         |0        0
MinFrameLength         |0        0
MplsFrameCount         |0        0
MplsFrameRate          |0        0
OversizeFrameCount     |0        0
OversizeFrameRate      |0        0
PauseFrameCount        |0        0
PauseFrameRate         |0        0
PrbsBitErrorCount      |0        0
PrbsBitErrorRate       |0        0
PrbsBitErrorRatio      |0.000000 0.000000
PrbsFillOctetCount     |0        0
PrbsFillOctetRate      |0        0
SigFrameCount          |214141   252765
SigFrameRate           |0        0
TcpChecksumErrorCount  |0        0
TcpChecksumErrorRate   |0        0
TcpFrameCount          |0        0
TcpFrameRate           |0        0
TotalBitRate           |0        0
TotalFrameCount        |214141   252765
TotalFrameRate         |0        0
TotalOctetCount        |27410048 32353920
TotalOctetRate         |0        0
Trigger1Count          |0        0
Trigger1Name           |
Trigger1Rate           |0        0
Trigger2Count          |0        0
Trigger2Name           |
Trigger2Rate           |0        0
Trigger3Count          |0        0
Trigger3Name           |
Trigger3Rate           |0        0
Trigger4Count          |0        0
Trigger4Name           |
Trigger4Rate           |0        0
Trigger5Count          |0        0
Trigger5Name           |
Trigger5Rate           |0        0
Trigger6Count          |0        0
Trigger6Name           |
Trigger6Rate           |0        0
Trigger7Count          |0        0
Trigger7Name           |
Trigger7Rate           |0        0
Trigger8Count          |0        0
Trigger8Name           |
Trigger8Rate           |0        0
UdpChecksumErrorCount  |0        0
UdpChecksumErrorRate   |0        0
UdpFrameCount          |0        0
UdpFrameRate           |0        0
UndersizeFrameCount    |0        0
UndersizeFrameRate     |0        0
VlanFrameCount         |0        0
VlanFrameRate          |0        0

GeneratorPortResults
Port                        |1:3     1:4
```

```
------------------------------+-----------------
GeneratorAbortFrameCount      |0        0
GeneratorAbortFrameRate       |0        0
GeneratorBitRate              |0        0
GeneratorCrcErrorFrameCount   |0        0
GeneratorCrcErrorFrameRate    |0        0
GeneratorFrameCount           |295595   280750
GeneratorFrameRate            |0        0
GeneratorIpv4FrameCount       |295595   280750
GeneratorIpv4FrameRate        |0        0
GeneratorIpv6FrameCount       |0        0
GeneratorIpv6FrameRate        |0        0
GeneratorJumboFrameCount      |0        0
GeneratorJumboFrameRate       |0        0
GeneratorL3ChecksumErrorCount |0        0
GeneratorL3ChecksumErrorRate  |0        0
GeneratorL4ChecksumErrorCount |0        0
GeneratorL4ChecksumErrorRate  |0        0
GeneratorMplsFrameCount       |0        0
GeneratorMplsFrameRate        |0        0
GeneratorOctetCount           |37836160 35936000
GeneratorOctetRate            |0        0
GeneratorOversizeFrameCount   |0        0
GeneratorOversizeFrameRate    |0        0
GeneratorSigFrameCount        |295595   280750
GeneratorSigFrameRate         |0        0
GeneratorUndersizeFrameCount  |0        0
GeneratorUndersizeFrameRate   |0        0
GeneratorVlanFrameCount       |0        0
GeneratorVlanFrameRate        |0        0
HwFrameCount                  |0        0
TotalBitRate                  |0        0
TotalFrameCount               |295595   280750
TotalFrameRate                |0        0
TotalIpv4FrameCount           |295595   280750
TotalIpv4FrameRate            |0        0
TotalIpv6FrameCount           |0        0
TotalIpv6FrameRate            |0        0
TotalMplsFrameCount           |0        0
TotalMplsFrameRate            |0        0
TotalOctetCount               |37836160 35936000
TotalOctetRate                |0        0

Spirent TestCenter>subscribe TxTrafficGroupResults
Subscribed on result type(s) TxTrafficGroupResults
Spirent TestCenter>transmit start
Successfully started transmit on port(s): 1:3,1:4
Spirent TestCenter>transmit stop
Successfully stopped transmit on port(s): 1:3,1:4
Spirent TestCenter>show results TxTrafficGroupResults
TrafficGroup |
Tg1                                 |
Tg2                                 |
-------------+-------------------------------------------------------------
---------+-------------------------------------------------------------
-----+
SubGroup     |
Sub1                                |
Sub2                          |Sub3   |
-------------+-------------------------------------------------------------
---------+-------------------------------------------------------------+--
```

```
-----+
StreamBlock  |           1:3/1            |           1:3/2
|1:4/1  |           1:3/1            |           1:3/2            |1:4/1
|
-------------+----------------------------+----------------------------
-+-------+----------------------------+----------------------------+--
-----+
StreamIndex  |0    1    2    3    4    |5    6    7    8    9
|0      |0    1    2    3    4    |5    6    7    8    9    |0      |
-------------+----------------------------+----------------------------
-+-------+----------------------------+----------------------------+--
-----+
BitRate      |0    0    0    0    0    |0    0    0    0    0
|0      |0    0    0    0    0    |0    0    0    0    0    |0      |
FrameCount   |66350 66350 66350 66350 66350 |66348 66348 66348 66348 66348
|637833 |66350 66350 66350 66350 66350 |66348 66348 66348 66348 66348 |637833 |
FrameRate    |0    0    0    0    0    |0    0    0    0    0
|0      |0    0    0    0    0    |0    0    0    0    0    |0      |
OctetCount   |0    0    0    0    0    |0    0    0    0    0
|0      |0    0    0    0    0    |0    0    0    0    0    |0      |
OctetRate    |0    0    0    0    0    |0    0    0    0    0
|0      |0    0    0    0    0    |0    0    0    0    0    |0      |
StreamId     |65536 65537 65538 65539 65540 |65541 65542 65543 65544 65545
|131072 |65536 65537 65538 65539 65540 |65541 65542 65543 65544 65545 |131072 |


Spirent TestCenter>stream disable 1:3 1
Successfully disabled stream 1 on port 1:3
Spirent TestCenter>show info stream-block
StreamBlock                      |1:3/1            1:3/2            1:4/1
---------------------------------+----------------------------------------
--------------
Active                           |false            true             true
BpsLoad                          |100000000.000000 100000000.000000
100000000.000000
BurstSize                        |1                1                1
ConstantFillPattern              |0                0                0
EnableBidirectionalTraffic       |false            false            false
EnableControlPlane               |false            false            false
EnableFcsErrorInsertion          |false            false            false
EnableStreamOnlyGeneration       |false            false            false
EnableTxPortSendingTrafficToSelf |false            false            false
EndpointMapping                  |ONE_TO_ONE       ONE_TO_ONE
ONE_TO_ONE
FillType                         |CONSTANT         CONSTANT
CONSTANT
FixedFrameLength                 |128              128              128
FlowCount                        |5                5                1
FpsLoad                          |84459.000000     84459.000000
84459.000000
FrameLengthMode                  |FIXED            FIXED            FIXED
IbgInMillisecondsLoad            |1344.000000      1344.000000
1344.000000
IbgInNanosecondsLoad             |1344.000000      1344.000000
1344.000000
IbgLoad                          |1344.000000      1344.000000
1344.000000
InsertSig                        |true             true             true
InterFrameGap                    |12               12               12
IsArpResolved                    |false            false            false
KbpsLoad                         |100000.000000    100000.000000
```

```
100000.000000
Load                            |10.000000         10.000000
10.000000
LoadUnit                        |PERCENT_LINE_RATE PERCENT_LINE_RATE
PERCENT_LINE_RATE
MaxFrameLength                  |256               256               256
MbpsLoad                        |100.000000        100.000000
100.000000
MinFrameLength                  |128               128               128
Name                            |StreamBlock 1     StreamBlock 7
StreamBlock 2
PercentageLoad                  |10.000000         10.000000
10.000000
Priority                        |0                 0                 0
RunningState                    |STOPPED           STOPPED
STOPPED
StartDelay                      |0                 0                 0
State                           |READY             READY             READY
Status                          |None              None              None
StepFrameLength                 |1                 1                 1
StreamBlockIndex                |0                 1                 0
StreamCount                     |5                 5                 1
StreamIndex                     |
Targets                         |ipv4:ipv41        ipv4:ipv42
ipv4:ipv43
TrafficPattern                  |BACKBONE          BACKBONE
BACKBONE
dstbinding-Targets              |ipv4if2           ipv4if2
ipv4if1
srcbinding-Targets              |ipv4if1           ipv4if1
ipv4if2

Spirent TestCenter>clear stats
Successfully cleared all statistics
Spirent TestCenter>unsubscribe all
Successfully unsubscribed from all result types
Spirent TestCenter>subscribe AnalyzerPortResults GeneratorPortResults
ArpNdResults
Subscribed on result type(s) AnalyzerPortResults, GeneratorPortResults,
ArpNdResults
Spirent TestCenter>sequencer step
Sequencer stepped successfully. Current command:
ArpNdStartOnAllStreamBlocksCommand 1
Spirent TestCenter>sequencer step
Sequencer stepped successfully. Current command: WaitCommand 1
Spirent TestCenter>sequencer run
Sequencer ran successfully
Spirent TestCenter>show stats port
AnalyzerPortResults
Port                    |1:3     1:4
------------------------+------------------
ComboTriggerCount       |0       0
ComboTriggerRate        |0       0
FcsErrorFrameCount      |0       0
FcsErrorFrameRate       |0       0
HwFrameCount            |0       0
IcmpFrameCount          |0       0
IcmpFrameRate           |0       0
Ipv4ChecksumErrorCount  |0       0
Ipv4ChecksumErrorRate   |0       0
Ipv4FrameCount          |337350  326285
```

```
Ipv4FrameRate          |0        0
Ipv6FrameCount         |0        0
Ipv6FrameRate          |0        0
Ipv6OverIpv4FrameCount |0        0
Ipv6OverIpv4FrameRate  |0        0
JumboFrameCount        |0        0
JumboFrameRate         |0        0
MaxFrameLength         |0        0
MinFrameLength         |0        0
MplsFrameCount         |0        0
MplsFrameRate          |0        0
OversizeFrameCount     |0        0
OversizeFrameRate      |0        0
PauseFrameCount        |0        0
PauseFrameRate         |0        0
PrbsBitErrorCount      |0        0
PrbsBitErrorRate       |0        0
PrbsBitErrorRatio      |0.000000 0.000000
PrbsFillOctetCount     |0        0
PrbsFillOctetRate      |0        0
SigFrameCount          |337350   326285
SigFrameRate           |0        0
TcpChecksumErrorCount  |0        0
TcpChecksumErrorRate   |0        0
TcpFrameCount          |0        0
TcpFrameRate           |0        0
TotalBitRate           |0        0
TotalFrameCount        |337354   326325
TotalFrameRate         |0        0
TotalOctetCount        |43181056 41767040
TotalOctetRate         |0        0
Trigger1Count          |0        0
Trigger1Name           |
Trigger1Rate           |0        0
Trigger2Count          |0        0
Trigger2Name           |
Trigger2Rate           |0        0
Trigger3Count          |0        0
Trigger3Name           |
Trigger3Rate           |0        0
Trigger4Count          |0        0
Trigger4Name           |
Trigger4Rate           |0        0
Trigger5Count          |0        0
Trigger5Name           |
Trigger5Rate           |0        0
Trigger6Count          |0        0
Trigger6Name           |
Trigger6Rate           |0        0
Trigger7Count          |0        0
Trigger7Name           |
Trigger7Rate           |0        0
Trigger8Count          |0        0
Trigger8Name           |
Trigger8Rate           |0        0
UdpChecksumErrorCount  |0        0
UdpChecksumErrorRate   |0        0
UdpFrameCount          |0        0
UdpFrameRate           |0        0
UndersizeFrameCount    |0        0
UndersizeFrameRate     |0        0
```

```
VlanFrameCount         |0        0
VlanFrameRate          |0        0

GeneratorPortResults
Port                           |1:3     1:4
-------------------------------+------------------
GeneratorAbortFrameCount       |0        0
GeneratorAbortFrameRate        |0        0
GeneratorBitRate               |0        0
GeneratorCrcErrorFrameCount    |0        0
GeneratorCrcErrorFrameRate     |0        0
GeneratorFrameCount            |326285   337350
GeneratorFrameRate             |0        0
GeneratorIpv4FrameCount        |326285   337350
GeneratorIpv4FrameRate         |0        0
GeneratorIpv6FrameCount        |0        0
GeneratorIpv6FrameRate         |0        0
GeneratorJumboFrameCount       |0        0
GeneratorJumboFrameRate        |0        0
GeneratorL3ChecksumErrorCount  |0        0
GeneratorL3ChecksumErrorRate   |0        0
GeneratorL4ChecksumErrorCount  |0        0
GeneratorL4ChecksumErrorRate   |0        0
GeneratorMplsFrameCount        |0        0
GeneratorMplsFrameRate         |0        0
GeneratorOctetCount            |41764480 43180800
GeneratorOctetRate             |0        0
GeneratorOversizeFrameCount    |0        0
GeneratorOversizeFrameRate     |0        0
GeneratorSigFrameCount         |326285   337350
GeneratorSigFrameRate          |0        0
GeneratorUndersizeFrameCount   |0        0
GeneratorUndersizeFrameRate    |0        0
GeneratorVlanFrameCount        |0        0
GeneratorVlanFrameRate         |0        0
HwFrameCount                   |0        0
TotalBitRate                   |0        0
TotalFrameCount                |326325   337354
TotalFrameRate                 |0        0
TotalIpv4FrameCount            |326285   337350
TotalIpv4FrameRate             |0        0
TotalIpv6FrameCount            |0        0
TotalIpv6FrameRate             |0        0
TotalMplsFrameCount            |0        0
TotalMplsFrameRate             |0        0
TotalOctetCount                |41767040 43181056
TotalOctetRate                 |0        0

ArpNdResults
Port                           |1:3 1:4
-------------------------------+--------
RxArpReplyCount                |0   0
RxArpRequestCount              |4   40
RxGratuitousArpCount           |0   0
RxIcmpV6ChecksumErrorCount     |0   0
RxIcmpV6CodeErrorCount         |0   0
RxIcmpV6LengthErrorCount       |0   0
RxIpV6HopLimitErrorCount       |0   0
RxIpV6PayloadLengthErrorCount  |0   0
RxNdDestAddrErrorCount         |0   0
RxNdOptionErrorCount           |0   0
```

```
RxNdTargetAddrErrorCount      |0   0
RxNeighborAdvertisementCount  |0   0
RxNeighborSolicitationCount   |0   0
TxArpReplyCount               |0   0
TxArpRequestCount             |40  4
TxNeighborAdvertisementCount  |0   0
TxNeighborSolicitationCount   |0   0

Spirent TestCenter>exit


Disconnected...


Press Enter to start a new session in this window...
```

Setting preferences for Spirent TestCenter CLI sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Spirent TestCenter.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Spirent TestCenter

Some session types (traffic generator types in particular) rely on a Tcl interpreter to do their work. In some cases you may need to control exactly which interpreter is used for a particular session type. You can specify the path to the correct interpreter in the session profile for the session, or if you prefer, you may set a preference for all sessions of a particular type.

For example, if TestCenter software needs a particular version of Tcl (which is not the same as the default interpreter specified in the system's PATH): The recommended practice is to specify the required interpreter in the session profile used to start the SmartBits session. By changing the preference, you can avoid changing the properties in the session profile, but your session profile may become less portable. If you send your test case to another user who does not have their preference configured the same way, then the session may fail.

Interpreter	Default: **Auto-select** **Note:** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: If an interpreter is specified in the Use the specified Tcl interpreter property, then use that interpreter. Otherwise, launch the first installed Tcl interpreter that iTest finds in the path environment variable. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter.
Log Tcl commands to a console	Check the box to log any Tcl commands to a console. Default: unchecked

Log Tcl responses to a console	Check the box to log all responses of the Tcl interpreter to a console. Default: unchecked
Remote shell logging	Default: unchecked Use remote shell logging.

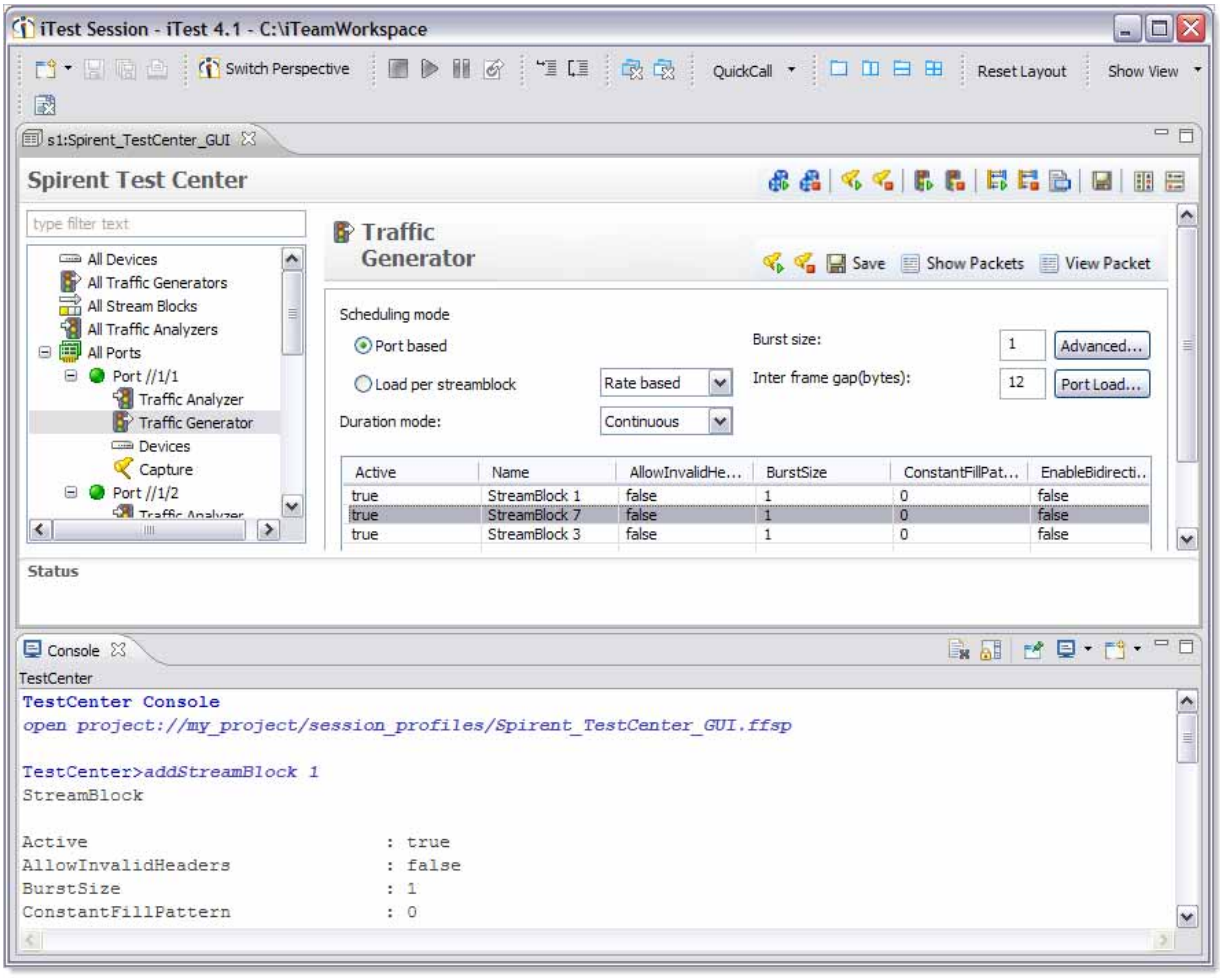
CHAPTER 53

Spirent TestCenter sessions

Spirent TestCenter session window

The session window for Spirent TestCenter sessions in iTest has been designed to closely resemble Spirent TestCenter. As a result, you can capture TestCenter steps using iTest without having to learn a new interface or new command names. You interact with the iTest session almost exactly like you interact with TestCenter.

Because all actions and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent TestCenter.

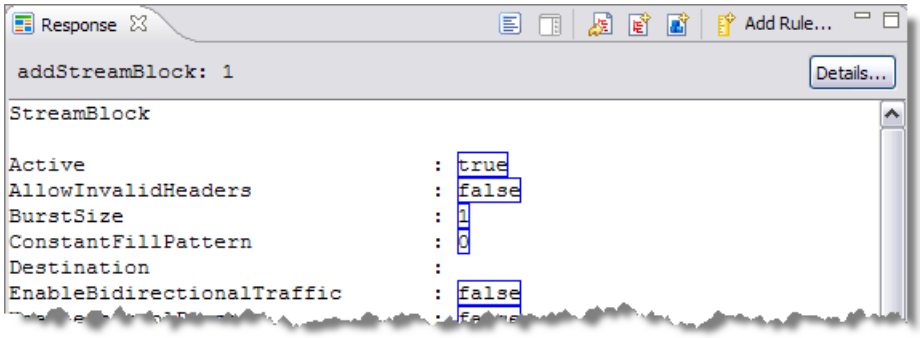


- Any action that you perform in the iTest session is forwarded to Spirent TestCenter running on the device. TestCenter performs the action and returns its normal text response. You can view the response in the **Results** section on the iTest window, just like you do in TestCenter.
- iTest captures all of the actions that you perform in a TestCenter session and all of the responses returned by TestCenter.

Responses are mapped

iTest does not use the TestCenter protocol to collect or analyze the traffic data—other steps in your test case can do that.

iTest saves all responses to TestCenter steps as structured data and auto-maps the responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. Here's the auto-mapped response to an **addStreamBlock** step in an iTest TestCenter session:



To use Demo mode

You have the option to run sessions in Demo mode, where iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. The intent is that you can run a session to learn more about how TestCenter sessions operate without having to install or run any Spirent software. See the descriptions of the **Demo mode** settings in the session profile: "Demo Mode" on page 1093.

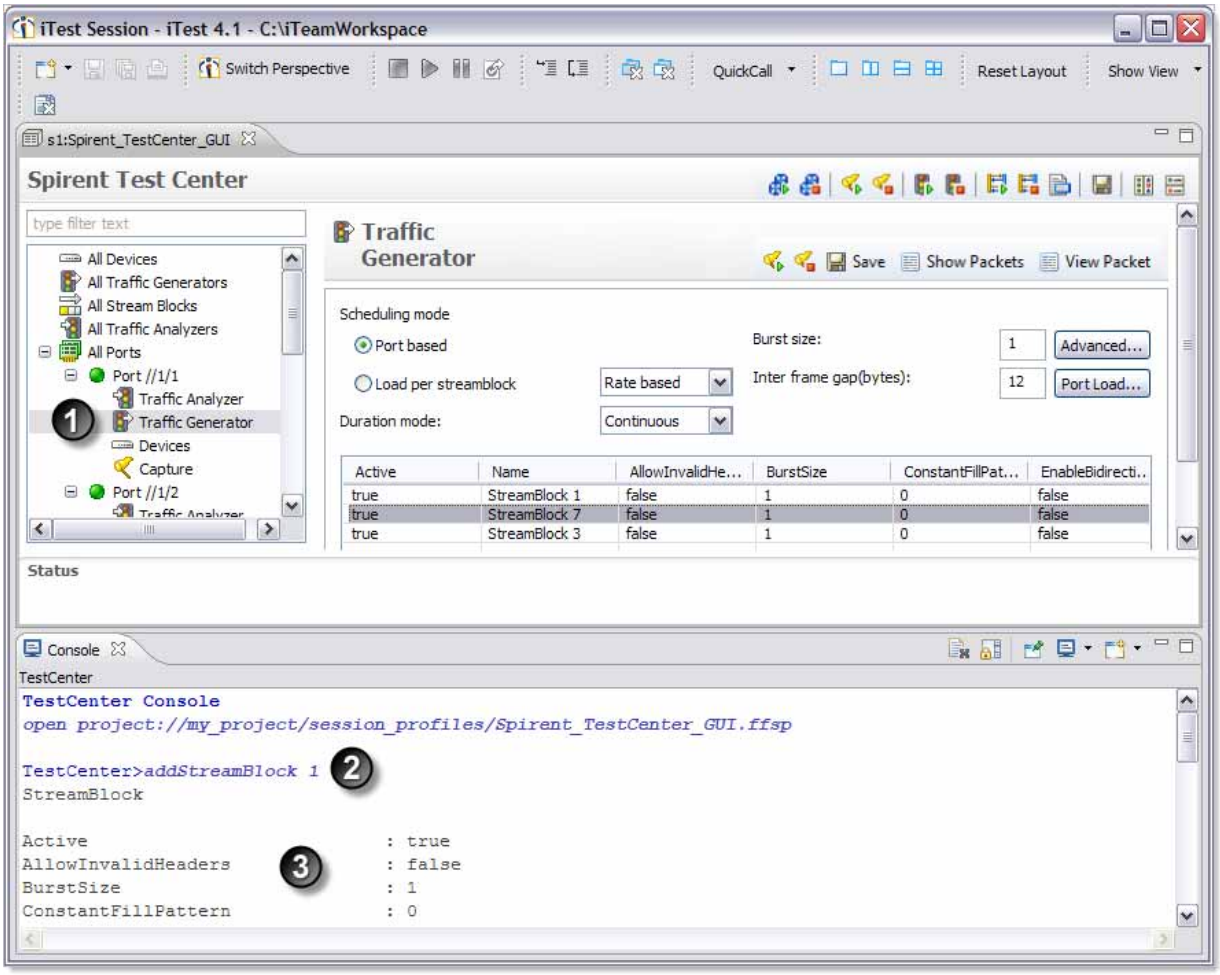
To create a test case that includes Spirent TestCenter sessions

You typically save captured manual steps as a test case. Follow these steps:

1 Ensure that the TestCenter session profile or device is properly configured. See "Session profile property settings for Spirent TestCenter sessions" on page 1091.

2 Click Start Capture to begin the direct-to-test process of saving the interactive session as a test case.

3 Start the TestCenter session and perform the test as needed. You work in the iTest TestCenter session the same way you work in TestCenter. When you interact with a TestCenter component, iTest performs a TestCenter action and captures both the action and the response from TestCenter. For example:



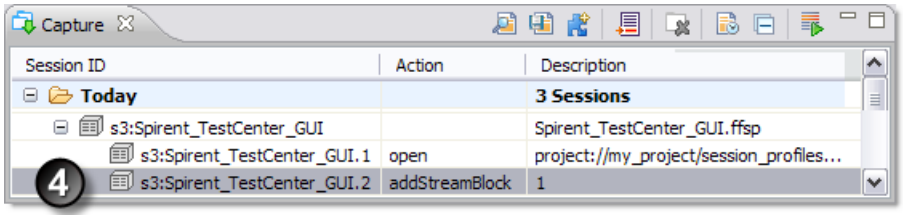
1 You first select Traffic Generator Port 1. When you click Add Raw Stream Block, iTest opens the **StreamBlock Editor** to enable you to configure the stream block (just like TestCenter). Then, when you click **OK** in the editor:

2 iTest submits an **addStreamBlock 1** command to TestCenter on the Spirent device (add a stream block on the generator's port 1).

3 In the session's **Console** window, iTest displays the **addStreamBlock 1** command. The response from TestCenter includes stream block settings that were implemented on the Spirent device.

Tip You can execute the Tcl **source** and **eval** actions and all STC commands at the command line.

Here is the captured item in the



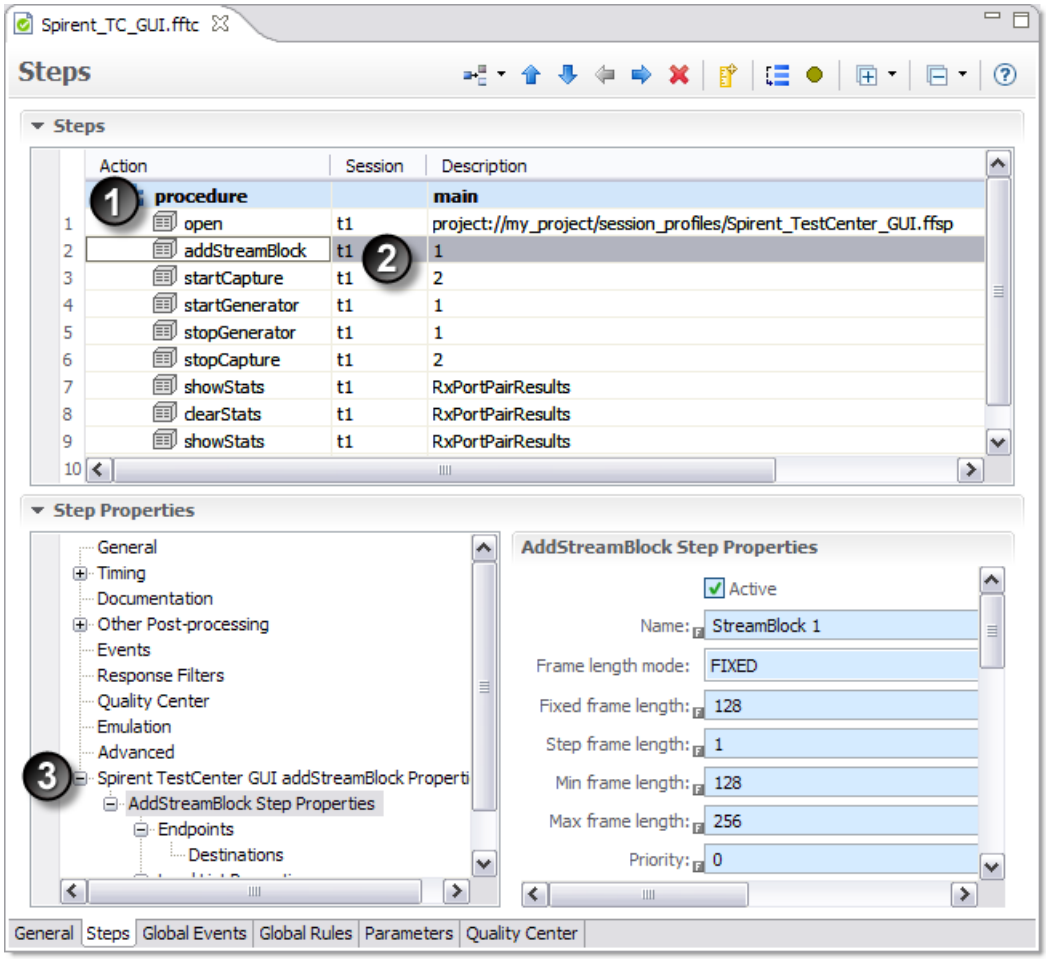
4 iTest captures an **addStreamBlock** step. Notice that the test case steps that iTest created from captured step refers to the port using the port ID (**1** in the example). You can configure iTest to create steps using a variety or port formats (see "To specify a list of port locations" on page 1050). The captured step includes the command that was sent to TestCenter on the Spirent device and its response. The captured step will become a step in the resulting test case, and it will execute exactly as you performed it in the interactive test.

4 Continue working in the interactive TestCenter session. iTest captures both commands and responses. (See "Spirent TestCenter result types in iTest" on page 1062 for details.)

Note While not all TestCenter commands are available using buttons or other controls on the page, you can perform any TestCenter command by entering it on the iTest Console view (as described in "Spirent TestCenter Command reference" on page 1048).

5 When you finish, click Finish Capture to save the captured steps into a test case. (The **Add Test Case** wizard opens and help you through the process. For details on saving captured steps to test cases, see "Saving interactive steps as steps in a test case: The 'Add to iTest Test Case' wizard" on page 98.)

6 The Test Case editor opens the new test case. Let's look at the steps in the test case and the properties of the example step.



1 *The open step for the session*

You opened a session and iTest auto-assigns the resulting **open** step a **session ID** of **t1**.

2 *The addStreamBlock step*

iTest generated this **addStreamBlock** step because it captured an **addStreamBlock** action when you clicked Add Raw Stream Block and clicked **OK** in the **StreamBlock Editor**. Notice that the **Description** field for the step specifies the port (**1**) to add the stream block to.

Note Remember that the device's response to this command is auto-mapped, so, you could add an analysis rule for the step to have the automated test verify the settings or extract a value from the response.

We captured a few steps after the **addStreamBlock** step. Let's look at the other steps in the test case. We can see that the test case will:

Step 3: Start capture on port 2 (we selected Traffic Generator Port 2 and then clicked)

Step 4: Start the generator on port 1 (we selected Traffic Generator Port 1 and then clicked)

Step 5: Stop the generator on port 1

Step 6: Stop capture on port 2

Step 7: Get the **RxPortPairResults** data for port 2 (in the **Statistics** portion of the tree for port 2, we selected **RxPortPairResults** and then clicked Show Stats. iTest captured the action and returned the data. You could, for example, add an analysis rule here to verify the data that was returned.)

Step 8: Clear the **RxPortPairResults** statistics for port 2

Step 9: Get the **RxPortPairResults** statistics for port 2 again (this step exists so you can add analysis rules to verify that step 8 indeed cleared the starts)

Step properties for the addStreamBlock step

In addition to the port number for the **addStreamBlock** step, iTest captured all of the settings that you configured in the TestCenter **StreamBlock Editor** when you submitted the step. You can view (and edit) all of the captured property settings in the **Step Properties** section for the step.

In the example, we selected the **addStreamBlock Step Properties** for the step. The settings here are identical to the settings that you make on the **StreamBlock Editor** while configuring a stream block.

7 Continue editing the test case by adding analysis rules and flow-control steps as needed.

- For information on working with test cases, see Chapter 7, "Test Cases".
- For information on using the Test Case editor, see Chapter 8, "Test Case Editor".

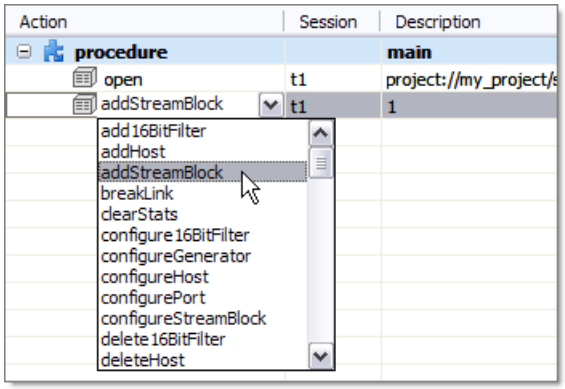
Spirent TestCenter Command reference

This topic describes all TestCenter commands in a session with iTest. The actions are captured during interactive STC sessions and you can add actions as described in "To add a TestCenter step to a test case" on page 1049.

iTest TestCenter sessions support all commands that are generated by UI elements, **eval** and **source** commands, and all TestCenter **sequencer** commands.

To add a TestCenter step to a test case

For a step in a TestCenter session, click in the **Action** cell and select the action from the drop-down list.



Slot and port numbering

- Slot and port numbering starts with 1 (1-based).
- When iTest returns responses to commands or captures your commands, it can display port references either in **slot:portID** format (the full port name, for example **2:3**) or in sequential **portIndex** format (just the port identifier, for example, **5**). The port format that is displayed in responses is controlled by the **Display slot:port format in responses** property. The port format used in captured steps is controlled by the **Use slot:port format in captured steps** property. See "Spirent TestCenter properties" on page 1091.

To specify a list of port locations

When specifying a list of ports, separate port identifiers using commas or space characters.

Use any of the following formats (and any mix of the formats) to specify a list of ports in a command (typically using a *portList* argument) and in the **Ports** session profile property setting.

Use iTest to generate a list with a large number of ports, for example:

- **slot:port** format — For example, **1:2 1:3** refers to Ports **2** and **3** in slot **1** of the Chassis specified by the **Chassis IP** property

  Note **slot:port** notation is valid only if the **Chassis IP** property value is valid and non-blank

- **//chassis/slot/port** format — For example, **//6.7.8.9/1/2 //6.7.8.9/1/3**

  refers to Ports **2** and **3** in Slot **1** of the Chassis at **6.7.8.9**

- **portIndex** format — The port identifier alone. For example, **2** refers to the second port in the model.
- **port handle** format — This format eases compatibility with sequencer and with STC commands that return handles. For example, **port1** refers to the first port that received a handle.

  Note Do not hard-code handles in a test case.

- **DDN** format — Of the form **project1.Port.portNumber**

  For simplicity, you can leave off the **project1** part of the notation and it will be assumed because most test cases have only one project. For example, **project1.Port.2** and **Port.2** refer to the second port in the model, and are equivalent to **2** of portIndex notation. Case is not significant in DDN notation.

To generate a port list for a large number of ports

Use iTest to generate a list with a large number of ports. For example:

- A loop can crawl the available chassis and build a list of the first **n** available ports, where **n** is the number of ports needed.
- **Velocity** can provide a list of ports.
- The topology associated with the test case contains port information. You can extract the information to build a port list.
- Store available port information in a file. A test case can read the file and build a port list.

Link aggregation group (LAG) aggregator port

Spirent TestCenter allows you to aggregate several physical ports into a link aggregation group (LAG) aggregator port. A LAG port is used much like a physical port. You can configure devices, protocols, and traffic on a LAG port as you would on a physical port. Traffic is sent and received across the ports in the LAG. Capture and traffic analysis is performed at the LAG level on a LAG port.

Note You can set up a link aggregation group (LAG) Offline or while you are connected to a chassis. If a LAG is set up Offline, it is validated when it is brought Online.

LAG support requires ports to be in the same port group. Port Group information is available during Port reservation, Port Relocation/Mapping and Equipment Information.

The Port Group tab (in STC Window) lists all of the port groups in the corresponding chassis and test modules.

Note In iTest, you can load the configuration file with the LAG information and substitute fields as required.

Syntax conventions Shown in the table below.

Convention	Description
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Arguments that appear within square brackets are optional.

Port commands

Action	Arguments / Command property values	Button that captures the Action	Description
breakLink	*portIndex*		Breaks the link on the selected port
configurePort	*portIndex*		Applies any changes that you made to configuration settings for the selected port. **Note** iTest supports Ethernet, 10G, 40G, 100G, and iEE802.11 wireless traffic. However, **iTest Spirent TestCenter session editor** allows you to view and configure port settings of only **Ethernet** and **IEE802.11 wireless traffic port**. You may view and configure all port settings in **TestCenter**.
restartAutoNegotiation	*portIndex*	— None —	Restart the auto-negotiation process for the selected ports.
restoreLink	*portIndex*	— None —	Bring the selected ports online and reserve for testing.
showPorts	[*portList*]	— None —	Shows information about the selected ports **Note** Test supports Ethernet, 10G, 40G, 100G, and iEE802.11 wireless traffic. However, **iTest Spirent TestCenter session editor** allows you to view and configure port settings of only **Ethernet** and **IEE802.11 wireless traffic port**. You may view and configure all port settings in **TestCenter**.

addPorts	*addPorts*	— None —	Add offline ports. In the Command property, specify portList which is the list of //chassis/slot/port separated by spaces.
attachPorts	*attachPorts*	— None —	Connects and reserves all ports that are not reserved yet.
mapPorts	*portIndex* and *portLocationLis*t		Maps ports to new locations. In the Command property, specify *portIndex* and *portLocationLis*t. *portIndex* is the first port to map, ports are mapped in order after that. Example: [mapPorts 2 //10.1.2.5/1/1 //10.1.2.5/1/2] will map Port 2 to //10.1.2.5/1/1 and Port 3 to //10.1.2.5/1/2. If ports were reserved, they should be unreserved first then use *attachPorts* to reserve again.
StartLagCaptureCommand	*PortSet(LAG port)*		Starts capture on the individual member ports of the LAG.
StopLagCaptureCommand	*PortSet(LAG port)*		Stops capture on the individual member ports of the LAG.

Traffic commands

Generator commands

Action	Arguments / Command property values	Button that captures the Action	Description
configureGenerator	*portIndex*	Apply	Applies any changes that you made to configuration settings
showGenerators	[*portList*]	— None —	Gets the property settings of the selected generator
startGenerator	[*portList*]	Selected port Generator start All ports	Starts transmitting on all or selected ports
stopGenerator	[*portList*]	Selected port Generator sto All ports	Stops transmitting on all or selected ports

Stream block commands

Limitation

When generating traffic, stream block will not work if it is inactive.

Action	Arguments / Command property values	Button that captures the Action	Description
addStreamBlock	*portIndex*	Add Raw Stream Block	Adds a raw stream block on the selected port.
configureStreamBlock	*portIndex streamIndex*	Edit...	Enables or disables transmission on a specified stream for selected ports Select a stream block and click **Edit**. When you click OK in the editor, iTest captures a configure action.
deleteStreamBlock	*portIndex streamIndex*	Delete	Deletes the selected stream block
disableStreamBlock	*portIdentifier* [*listOfStreamBlockIDs*]	Appears in local toolbar on **Traffic Generator** page	Disables the specified stream blocks on the specified port. You can specify *listOfStreamBlockIDs* using any combination of stream block IDs and ranges, separated by commas. For example, **1, 3-5, 7**

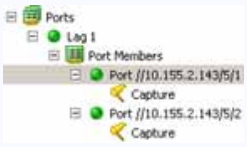
enableStreamBlock	*portIdentifier* [*listOfStreamBlockIDs*]	Appears in local toolbar on **Traffic Generator** page	Enables the specified stream blocks on the specified port. You can specify *listOfStreamBlockIDs* using any combination of stream block IDs and ranges, separated by commas or spaces. For example, **1, 3-5, 7**
showStreamBlocks	[*portList*]	Show Stream Blocks	Gets the configuration property settings for all stream blocks.
showStreamBlocksInGroup	*group* [*subgroup*]	Show	Gets the configuration property settings for all stream blocks in the selected Traffic Group.

Analyzer commands

Action	Arguments / Command property values	Button that captures the Action	Description
add16BitFilter	*portIndex*		Adds a custom 16-bit filter on the selected port.
configure16BitFilter	*portIndex analyzer16bitFilterIndex*	Apply	Applies any changes that you made to configuration settings
show16BitFilters	[*portList*]	— None —	Returns all information related to the 16-bit filter
showAnalyzers	[*portList*]	— None —	Gets the property settings of the selected analyzer

Capture commands

Action	Arguments / Command property values	Button that captures the Action	Description
saveCapture	*portIndex*	— None —	Saves the captured items
showPacket	*portIndex pktNumber*	View Packet	Returns the specified captured packet on the selected port
showPacketList	*port startPkt pktCount*	Show Packets	Returns information about the specified list of captured packets in table form
startCapture	[*portList*]	Selected port Capture start All ports	Starts capturing on all or selected ports
stopCapture	[*portList*]	Selected port Capture stop All ports	Stops capture on all or selected ports.

StartLagCaptureCommand	*PortSet[LAG port].*	Selected port  All ports	Starts capture on the individual member ports of the LAG.
StopLagCaptureCommand	*PortSet[LAG port].*	Same as above	Stops capture on the individual member ports of the LAG.
GetLagCaptureCommand	*PortSet(LAG port)*	Same as above	Merges capture of the individual member ports of the LAG.

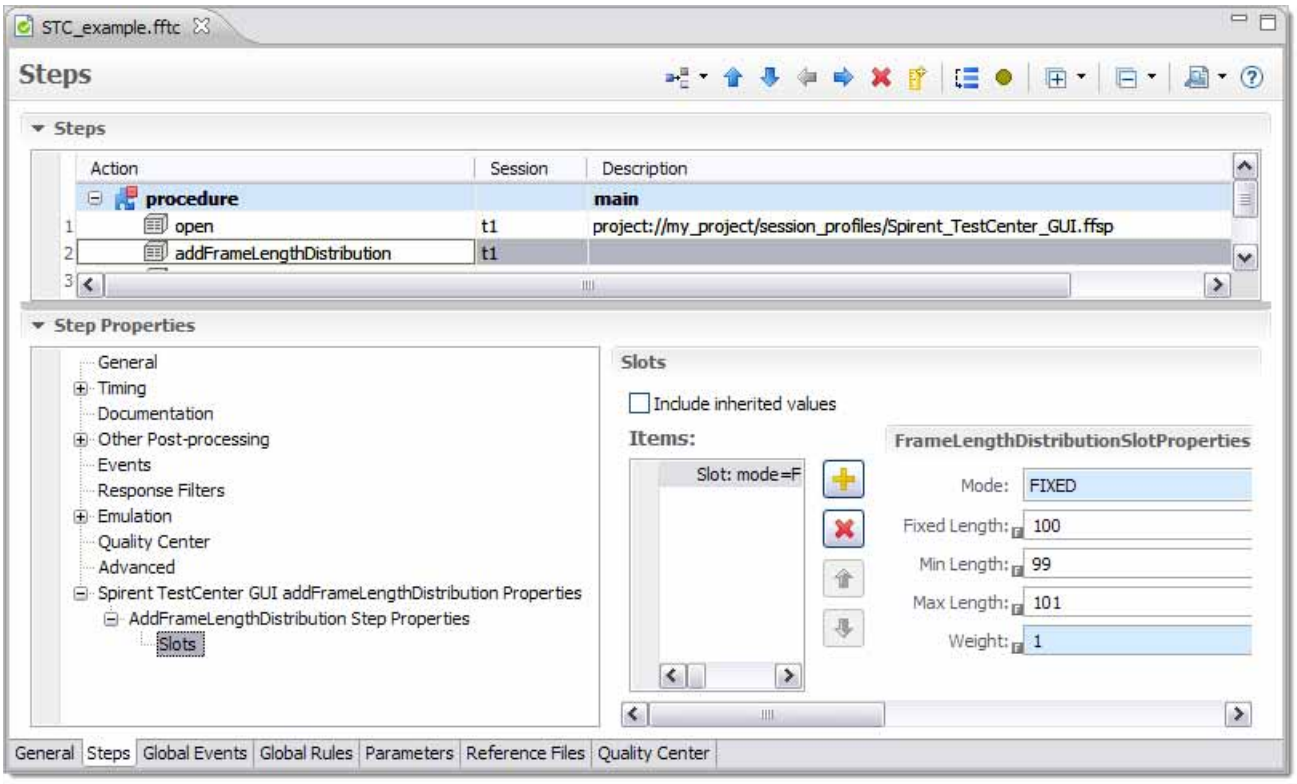
iMIX commands

Note For details on using the iMIX editor in interactive sessions and to see the commands that are captured when you click editor buttons, see "To edit iMIX settings" on page 1089.

Action	Arguments / Command property values	Button that captures the Action	Description
showFrameLengthDistributions	*distributionName*		Shows iMIX frame length distribution. See "Property settings" on page 1056
addFrameLengthDistribution	*distributionName*	See note above the table	Adds a new iMIX frame length distribution. See "Property settings" on page 1056.
configureFrameLengthDistribution	*distributionName*	See note above the table	Changes the settings for the specified frame length distribution See "Property settings" on page 1056.
deleteFrameLengthDistribution	*distributionName*	See note above the table	Deletes the specified frame length distribution.

Property settings

The **addFrameLengthDistribution** and **addFrameLengthDistribution** commands capture the following property settings (in the Step Properties section for the step as shown in the example).



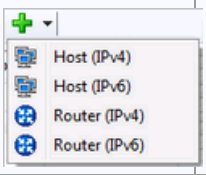
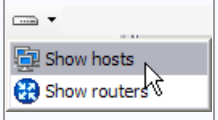
Name	Distribution name
Seed	Distribution seed
Slots — Each slot defines a concrete length distribution	
Mode	Slot mode: random or fixed
Fixed length	**FIXED** mode only: Fixed frame length
Min / Max length	**RANDOM** mode only: Minimum and maximum frame length
Weight	Slot weight

Example response for configureFrameLengthDistribution

```
TestCenter>configureFrameLengthDistribution JMIX Downstream
FrameLengthDistribution

Name : JMIX Downstream
Seed : 10900842

Mode  FixedLength MinLength    MaxLength     Weight
-------------------------------------------------
FIXED 60          59           61            3
FIXED 120         119          121           4
FIXED 576         575          577           1
FIXED 1500        1499         1500          5
FIXED 100         99           101           1
FIXED 100         99           101           1
FIXED 100         99           101           1
```

Device commands

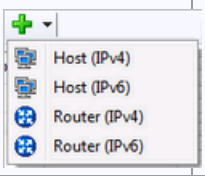
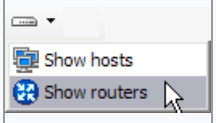
Host commands

Action	Arguments / Command property values	Button that captures the Action	Description
addHost	*portIndex*		Adds a new device to the list with role=Host . Only "Traffic Only" hosts are supported.
configureHost	*portIndex deviceIndex*	— None —	Applies any changes that you made to configuration settings *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **configureDevice** and **configureRouter** commands.
deleteHost	*portIndex deviceIndex*	— None —	Deletes the specified host. *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **deleteDevice** and **deleteRouter** commands.
showHosts	[*portList*]		Shows information about the 'host' parent object. The button returns information for all devices with role=Host. The **showHosts** action returns information for devices with role=Host on the specified ports. If no ports are specified, returns information for all devices with role=Host..

Device commands

Action	Arguments / Command property values	Button that captures the Action	Description
configureDevice	*portIndex deviceIndex*		Applies any changes that you made to configuration settings *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **configureRouter** and **configureHost** commands.
deleteDevice	*portIndex deviceIndex*		Deletes the specified devices. *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **deleteRouter** and **deleteHost** commands.
showDevices	[*portList*]		The button returns information for all devices. The **showDevices** action returns information for devices on the specified ports. If no ports are specified, returns information for all devices.
startDevices	[*portList*]		Starts the specified devices on the specified ports. If no ports are specified, then all devices are started. The button in the main frame starts all devices. The button on a port frame starts devices on that port.
stopDevices	[*portList*]		Stops the specified devices on the specified ports. If no ports are specified, then all devices are stopped. The button in the main frame stops all devices. The button on a port frame stops devices on that port.

Router commands

Action	Arguments / Command property values	Button that captures the Action	Description
addRouter	*portIndex*		Adds a new device to the list with role=Router
configureRouter	*portIndex deviceIndex*	— None —	Applies any changes that you made to configuration settings *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **configureDevice** and **configureHost** commands.
deleteRouter	*portIndex deviceIndex*	— None —	Deletes the specified host. *deviceIndex* is the number shown in the device table as **ID**. **Note** The command works identically as the **deleteDevice** and **deleteHost** commands.
showRouters	[*portList*]		The button returns information for all devices with role=Router. The **showRouters** action returns information for devices with role=Router on the specified ports. If no ports are specified, returns information for all devices with role=Router..

ARP packet actions

For all of the following ARP actions:

- Click the button that captures the action in the main toolbar to apply the action to all ports. Click the button on the "local" toolbar (on pages that enable you to select ports) to apply the action to the selected ports.
- You can specify the *listOfPortIdentifiers* argument values in any mix of formats separated by spaces. For example, **slot:port** mixed with sequential **portIndex — 1:1 1:2 3 6**

  See "To specify a list of port locations" on page 1050.
- If *listOfPortIdentifiers* is not specified, then the command is applied to all ports and returns data for all ports.

- The returned ARP/ND state is one of the following:

  IDLE — Idle

  WAITING — ARP/ND is in progress

  SUCCESSFUL — All attempted ARP/NDs were resolved successfully

  FAILURE — Some attempted ARP/NDs could not be resolved successfully

  CONGESTED — Some attempted ARP/NDs are congested

Action	Arguments / Command property values	Button that captures the Action	Description
startArpNd	[*listOfPortIdentifiers*]	 Appears in main toolbar for all ports and local toolbar for **Port** page	Starts ARP/ND packets for the specified ports and returns the ARP/ND state. Example: startArpNd 1:1 1:2 3
stopArpNd	[*listOfPortIdentifiers*]	 Appears in main toolbar for all ports and local toolbar for **Port** page	Stops ARP/ND packets for the specified ports and returns the ARP/ND state. Returns data for all ports if *listOfPortIdentifiers* is not specified. Example: stopArpNd 2 3 1:1
startArpNdOnAllStreamBlocks	[*listOfPortIdentifiers*]	 Appears in local toolbar on **All Stream Blocks** and **Traffic Generator** pages	Starts ARP/ND packets on all stream blocks for the specified ports and returns the ARP/ND state. Returns data for all ports if *listOfPortIdentifiers* is not specified. Example: startArpNdOnAllStreamBlocks 1:3 1:4
startArpNdOnAllDevices	[*listOfPortIdentifiers*]	 Appears in local toolbar on **All Devices** and **Devices** pages	Starts ARP/ND packets on all devices for the specified ports and returns the ARP/ND state. Returns data for all ports if *listOfPortIdentifiers* is not specified. Example: startArpNdOnAllDevices 1:3 1:4
showArpNdCache	[*listOfPortIdentifiers*]	 Appears in main toolbar for all ports and local toolbar for **Port** page	Returns the ARP/ND cache. See "Example response for showArpNdCache" on page 1061. Returns data for all ports if *listOfPortIdentifiers* is not specified. Example: showArpNdCache 1:3 1:4

Example response for showArpNdCache

```
Arp Cache:
Port  Stream / Router / Host  SourceIPAddress GatewayIPAddress ResolvedMacAddress
-------------------------------------------------------------------------------

Failed ARP Items:
Port  Stream / Router / Host  SourceIPAddress GatewayIPAddress ResolvedMacAddress
-------------------------------------------------------------------------------
//1/1 StreamBlock 14: 0     192.85.1.2    192.85.1.1     00:00:00:00:00:00
//1/2 StreamBlock 15: 0     192.85.1.2    192.85.1.1     00:00:00:00:00:00
```

Results commands

Result View commands

Action	Arguments / Command property values	Button that captures the Action	Description
clearResults	— None —	Clear Stats	Clears all statistics for all ports
saveResults	*resultType*	— None —	Shows the specified statistics for the 'port' parent object for the selected port. The stat must be subscribed.
showResults	*resultType*	— None —	Shows the specified statistics for the 'port' parent object for the selected port. The stat must be subscribed.
showSubscriptionViews	— None —	— None —	Subscribes for a specified result type or lists all available result types
subscribeView	*resultTypeList*	— None —	Subscribes for the specified result types.
unsubscribeView	*resultTypeList*	— None —	Unsubscribes for the specified result types

Spirent TestCenter result types in iTest

This table presents the native TestCenter result views that you should now use instead of the (now deprecated) iTest statistic result types. Spirent strongly recommends that you that you use TestCenter result views rather than statistics.

Note The list of result types is based on the installed version of TestCenter and so is subject to change as new TestCenter versions are released.

			Do not use:	**Use this instead:**	
Native TestCenter UI Grouping / Location			**Statistic Type (Deprecated — Not supported in iTest 4.3 and later)**	**iTest Result View (New in iTest 4.3)**	
				iTest STC Group	**Results View**
Port Traffic	Basic Traffic Results		AnalyzerPortResults, GeneratorPortResults	Port Traffic	Basic Traffic Results
	Diffserv Results		DiffServResults		Diffserv Results
	Port Average Latency Results		PortAvgLatencyResults		Port Average Latency Results
	Overflow Results		OverflowResults		Overflow Results
	Port Pair Results		Tx/RxPortPairResults		Port Pair Results
	CPU Port Results		Tx/RxCpuPortResults		CPU Port Results

Stream Results	Traffic Group Results	Tx/RxTrafficGroupResults	Stream Results	Traffic Group Results
	Filtered Stream Results	FilteredStreamResults		Filtered Stream Results
	Detailed Stream Results	TxStreamResults, RxStreamSummaryResults		Detailed Stream Results
	Stream Block Results	Tx/RxStreamBlockResults		Stream Block Results
Port Protocols	SONET Interface Results	SonetResults	Port Protocols	SONET Interface Results
	POS Interface Results	PppProtocolResults		POS Interface Results
	ARPND Results	ArpNdResults		ARPND Results
	LACP Results	LacpPortResults		LACP Results
	L2TP Results	L2tpPortResults	Access Protocols	L2TP Results
	PPPoX Results	PPPoEPortResults		PPPoX Results
	DHCP Results	Dhcpv4PortResults		DHCP Results
	DHCPv6 Results	Dhcpv6PortResults		DHCPv6PD Results
	IGMP Results	IgmpPortResults	Port Protocols	IGMP Results
	MLD Results	MldPortResults		MLD Results
	EOAM Results	EoamPortResults		EOAM Results

Router Protocols	BGP Results		BgpRouterResults	Routing Protocols	BGP Results
	OSPFv2 Results		Ospfv2Results		OSPFv2 Results
	OSPFv3 Results		Ospfv3Results		OSPFv3 Results
	ISIS Results		IsisRouterResults		ISIS Results
	RIP Results		RipRouterResults		RIP Results
	LDP Results		LdpRouterResults		LDP Results
	RSVP Results		RsvpRouterResults		RSVP Results
	PIM Results		PimRouterResults		PIM Results
	IGMP Querier Results		IgmpRouterResults		IGMP Querier Results
	MLD Querier Results		MldRouterResults		MLD Querier Results
	STP Results		BridgePortResults		STP Results
	MSTI Results		BridgePortResults		MSTI Results
	LDP-RSVP LSP Results	LDP LSP Results	LdpLspResults		LDP LSP Results
		RSVP LSP Results	RsvpLspResults		RSVP LSP Results
	EOAM Results	Port Results	EoamPortResults	Carrier Ethernet	Port Results
		CC Results	EoamContChkLocalResults		CC Results
		LT Results	EoamLinkTraceResults		LT Results
		MEG Results	EoamMegResults	Routing Protocols	MEG Results
		LB Results	EoamLoopbackResults		LB Results
Host Protocols	DHCP Results		Dhcpv4BlockResults	Access Protocols	DHCP Results
	PPPoX Results		PPP/PPPoeClientBlockResults, PPP/PPPoeServerBlockResults		PPPoX Results
	DHCPv6PD Results		Dhcpv6BlockResults		DHCPv6PD Results
	L2TP Results		L2TPv2BlockResults		L2TP Results
	IGMP Results		IgmpHostResults	Port Protocols	IGMP Results
	MLD Results		MldHostResults		MLD Results
	IGMP-MLD Group Results	IGMP Host-Group Results	IgmpGroupMembershipResults	Access Protocols	IGMP Host-Group Results
		MLD Host-Group Results	MldGroupMembershipResults		MLD Host-Group Results
	SIP Results		SipUaBlockResults	Application Layer Protocols	SIP Ua Results

IPTV	Test Results	IptvTestResults	IPTV	Test Results
	Port Results	IptvPortResults	Carrier Ethernet	Port Results
	STB Block Results	IptvStbBlockResults	IPTV	STB Block Results
	Viewing Profile Results	IptvViewingProfileResults		Viewing Profile Results
	Channel Results	IptvChannelResults		Channel Results

Statistics commands (Deprecated and not recommended)

Important The commands in the following table are deprecated and are described here only for backward compatibility with older versions. Spirent recommends that you do not use the commands. See "Spirent TestCenter result types in iTest" on page 1062 for details on the TestCenter GUI response views commands that you should use instead

Action	Arguments / Command property values	Description
clearStats	— None —	Clears all statistics for all ports
showStats	*resultType*	Shows the specified statistics for the 'port' parent object for the selected port. The stat must be subscribed.
showSubscription	— None —	Subscribes for a specified result type or lists all available result types
subscribe	*resultTypeList*	Subscribes for the specified result types.
unsubscribe	*resultTypeList*	Unsubscribes for the specified result types

General commands

Action	Arguments / Command property values	Button that captures the Action	Description
eval	*Tcl_statement*	— None —	Evaluates a Tcl statement for direct access to Spirent TestCenter's TCL API.

saveConfiguration	*filename*		Saves the current configuration information returned by the device into a specified file for all or selected ports. **Note** Spirent limits the path to 256 characters. The file is used to configure the device exactly as if you had configured it using TestCenter: In test cases, you use **configuration load** to load the configuration settings to the device. You can specify the resulting file in the **Configuration file** setting in a session profile. The configuration is performed when the session starts. The configuration file is saved in XML format. with filename extension **.xml**, Argument *filename* is the configuration file to save the config to Example: **project:///spirent.xml**
showTrafficGroups	— None —	— None —	Shows information about traffic groups
source	*Tcl_script*	— None —	Sources the specified TCL script from workspace

Sequencer control commands

Sequencer control commands

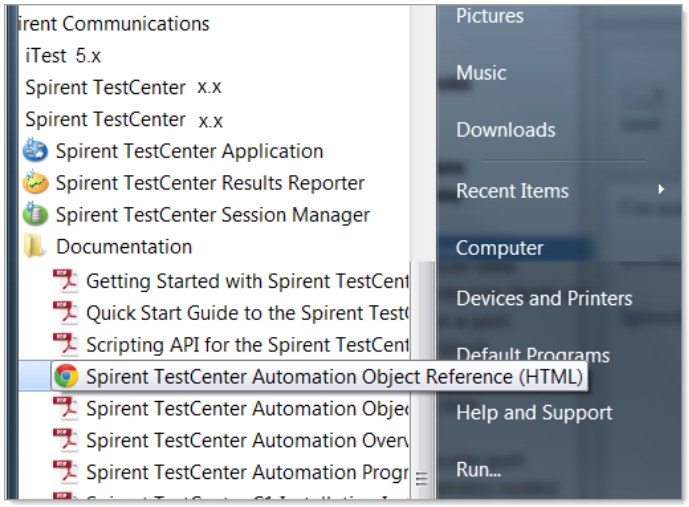
Action	Arguments / Command property values	Button that captures the Action	Description
pauseSequencer	— None —		Works like the STC API **SequencerPause** command: Pauses the sequencer. If you execute **pauseSequencer** while a command is running, the sequencer will pause after the command has finished. While the sequencer is paused and you invoke **stepSequencer** or **startSequencer**, Spirent TestCenter resumes command execution from the place that execution was paused.
runSequencer	— None —	— None —	**Note** See the **startSequencer** command below. Runs all the steps to end. The session is blocked in TestCenter console and GUI when executing **runSequencer** command. After the sequencer is completed, all steps are not displayed on the console. To display all sequencer steps, execute the **showSequencer** command.
showSequencer	— None —	— None —	Returns the current state of the selected sequencer. Possible values: IDLE, WAIT, and PAUSE
startSequencer	— None —		Gets the sequencer and issues the commands to chassis to start the sequencer. Immediately returns control to the user.
stepSequencer	— None —		Executes the next step in the sequence.
stopSequencer	— None —		Stops the current sequence execution.
waitSequencer	— None —		Waits for the current step in the current sequence to finish. Blocks the call until step is completed.

Sequencer action commands

To access the full set of sequencer action commands for TestCenter, type **sequencer** at the Console command line (**sequencer** is a first-level child state of the Testcenter.Native state). The sequencer commands at the console are equivalent to the NTAF commands.

The NTAF commands include all of TestCenter sequencer commands plus **get**, **getObjects**, **getObjectInfo**, and **getRelatives**.

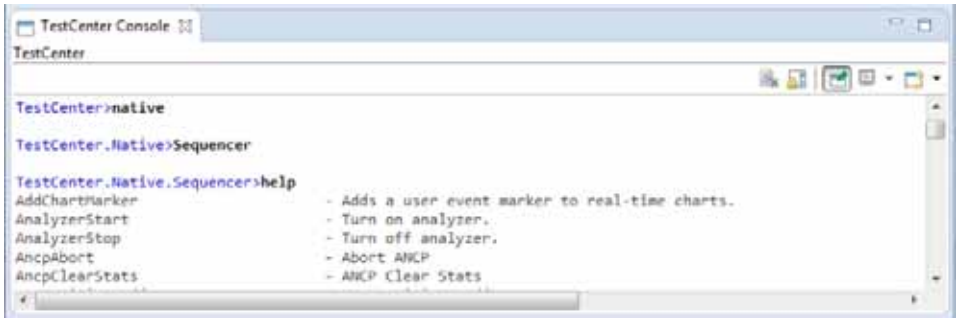
Note For a list of commands in iTest 5.n, see the "Command Sequencer Command Objects Index" and "All Command Objects Index" in the *Spirent TestCenter Automation Object Reference*.



- **Getting help: To view descriptions of sequencer control commands (including the 'get' family of commands) in iTest**

1. In the iTest TestCenter Console view, type **sequencer** to enter the sequencer state.
2. Type **help**

   The sequencer action commands are in **TestCenter.Native.Sequencer** as illustrated below.

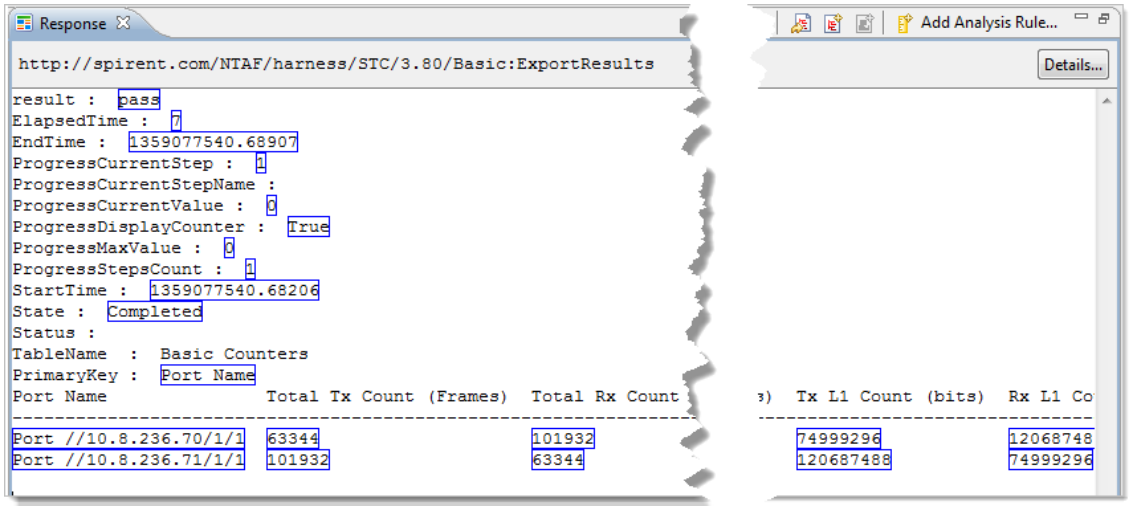


Responses to sequencer action commands

Most sequencer action commands return a list of attributes. The attributes map to a iTest response in the following form:

```
property1 : value1
property2 : value2
property3 : value3
```

The response to the **ExportResults** command includes the list of data plus data in a table:



Non-Sequencer action commands

To access the full set of non-sequencer action commands for TestCenter, type **nonsequencer** at the Console command line (**nonsequencer** is first-level child state of the Testcenter.Native state).

Note For a list of commands in iTest 5.n, see “All Command Objects Index” in the *Spirent TestCenter Automation Object Reference* guide.

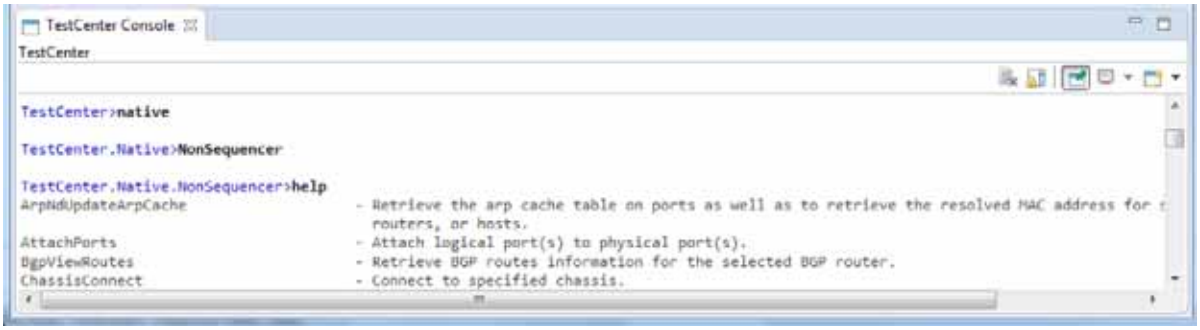
The **nonsequencer** commands are indicated as follows in the *Spirent TestCenter Automation Object Reference*:

Can be used with Command Sequencer: No

- **Getting help: To view descriptions of nonsequencer control commands in iTest**

1. In the iTest TestCenter Console view, type **NonSequencer** to enter the nonsequencer state.
2. Type **help**

The nonsequencer action commands are in **TestCenter.Native.NonSequencer** as illustrated below.



HLTAPI commands

To use the HLTAPI action commands in iTest, set up the HLTAPI package for TestCenter as described in "*Spirent TestCenter Automation: HLTAPI Environment Installation Guide*".

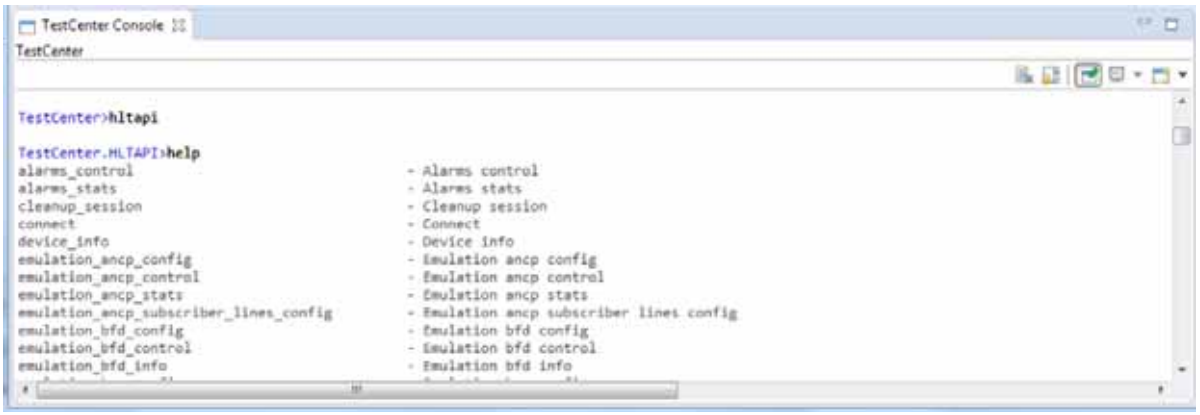
To access the complete list of HLTAPI action commands for TestCenter, type hltapi at console command line (hltapi is a first-level child state of the TestCenter when hltapi is enabled in TestCenter session).

Note For a list of HLTAPI commands in iTest 5.n, see *Spirent TestCenter Automation HLTAPI Command Reference*.

- **Getting help: To view descriptions of HLTAPI commands in iTest**

1. In the iTest TestCenter Console view, type **hltapi** to enter the hltapi state.
2. Type **help**

The hltapi action commands are in **TestCenter.hltapi** as illustrated below.



STAK commands

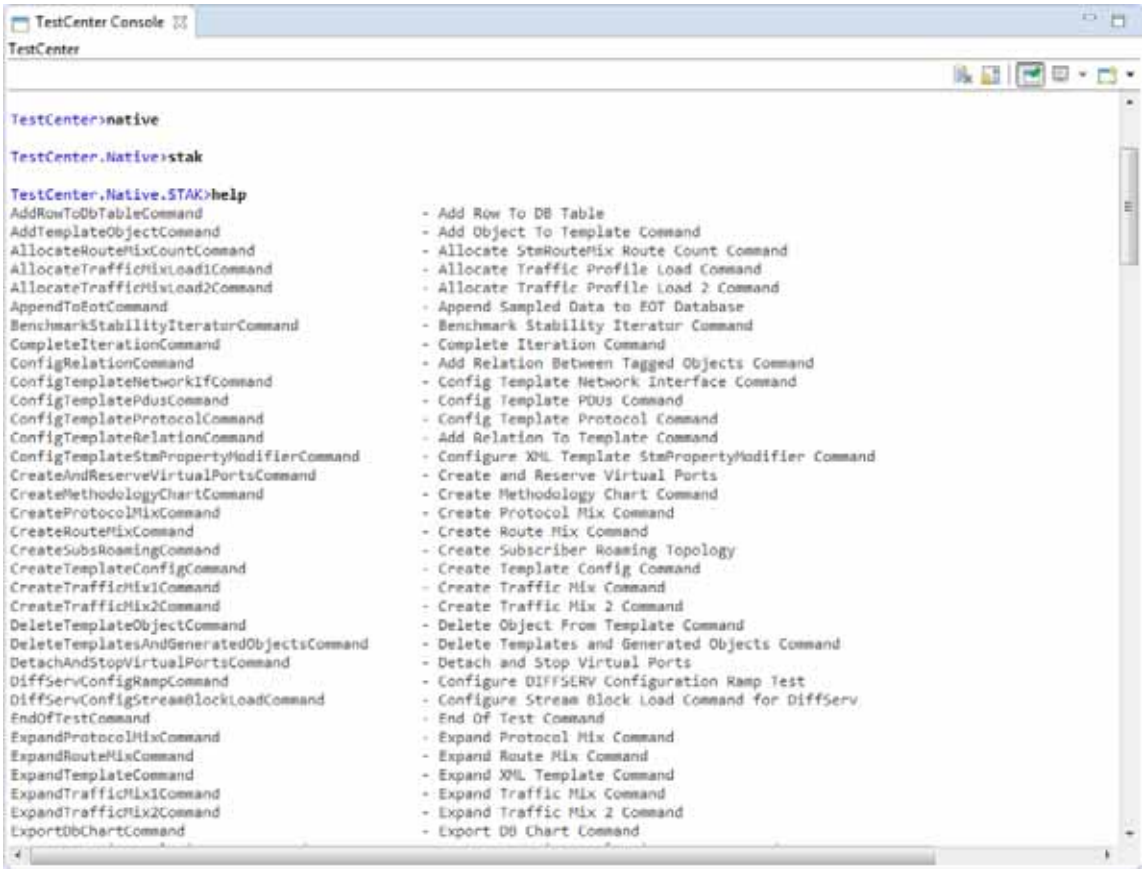
To access the full set of STAK action commands for TestCenter, type stak at console command line (stak is not a first-level child state of the TestCenter state, but is the first-level child state of the TestCenter.native state).

- **Getting help: To view descriptions of STAK commands in iTest**

1 In the iTest TestCenter Console view, type stak to enter the STAK state.

2 Type help

The stak action commands are in **TestCenter.Native.STAK** as illustrated below.



Note The STAK commands are available on the system running Spirent TestCenter when you search for commands.

Syntax: `help list commands stak`

To get a list of the STAK commands, you may also pass the stak keyword as the final parameter to the help list commands call. This lists all the STAK commands available on the Spirent TestCenter instance in use.

CLI integration commands

The TestCenter session type incorporates the following commands from the TestCenter CLI integration.

The TestCenter CLI integration is now **obsolete and is deprecated.**

capture save *port filename*	Saves capture data from the specified port to the specified file. **Note** Spirent limits the path to 256 characters.
capture start *[portList]*	Starts capturing on all or specified ports
capture stop *[portList]*	Stops capture on all or specified ports.
clear stats	Clears statistics for all ports
configuration load *filePath*	Configures the device with information from the specified file. **Note** Spirent limits the path to 256 characters. The **configuration load** command supports XML configuration files generated by **configuration save**. If the filename extension is **.xml**, then iTest interprets the configuration file as XML format. iTest expects that a second file exists in the same directory. The second file must have the same name postfixed with **_logic**. For example, **config.cfg** and **config.cfg_logic** During interactive TestCenter sessions, **configuration load** performs replacements on the file so that the specific information (like IP address) are translated into variables or command substitutions. Arguments Name Description Example filePath Configuration file to load project:///spirent.xml

configuration save *filePath*	Save the current configuration information returned by the device into a specified file. **Note** Spirent limits the path to 256 characters. The file is used to configure the device exactly as if you had configured it using TestCenter: In test cases, you use **configuration load** to load the configuration settings to the device. You can specify the resulting file in the **Configuration file** setting in a session profile. The configuration is performed when the session starts. The configuration file format depends on the filename extension of the specified file. If the extension is **.xml**, then the configuration is saved in XML format. If the target format is XML, you can provide the list of ports to include in the configuration file. Arguments Name Description Example filePath Configuration file to save the config to project:///spirent.xml
eval *Tcl_statement*	Evaluate a Tcl statement for direct access to Spirent TestCenter's TCL API
sequencer start	Gets the sequencer and issues the commands to chassis to start the sequencer. Immediately returns control to the user.
sequencer stop	Stops the current sequence execution.
sequencer step [-async]	Executes the next step in the sequence. If the **-async** option is provided, the step is started and the control is returned to the user immediately. In this case, use sequencer wait to wait for the step to finish.
sequencer wait	Waits for the current step in the named sequence to finish. Blocks the call until step is completed.
sequencer state	Returns the current state of the named sequence. Possible values: IDLE, WAIT, and PAUSE
sequencer run	Issues command to run all the steps to end. This command blocks until the sequence run is finished. It prints out a message for each step in the sequencer as the steps execute. If you press Ctrl-C, the sequencer run is aborted.
show info host	Shows information about the 'host' parent object
show info router	Shows information about the 'router' parent object. (Replaces **get routerinfo**)
show info stream-block *[portList]*	Shows streams from all or specified ports
show info *trafficGroup*	Shows information about traffic group
show packet *port packetIndex* [*-raw*]	Displays the specified captured packet on the specified port
show results *[objects]*	Shows results of specified type for all or specified objects

show stats *[portList]*	Displays a table of statistics for all ports or by port Shows subscribed stats for the 'port' parent object for the specified port list
source	Sources TCL script from workspace
stream enable *port streamIndex*	Enables transmission on a specified stream
stream disable *port streamIndex*	Disables transmission on a specified stream
subscribe	Subscribes for specific result type or lists all the result-types available
transmit start *[portList]*	Starts transmitting on all or specified ports
transmit stop *[portList]*	Stops transmitting on all or specified ports
unsubscribe	Unsubscribes for specific result type or lists all the result-types available
unsubscribe all	Unsubscribes for all available result types

Spirent TestCenter result types in iTest

This table correlates TestCenter data items with the responses to TestCenter commands in iTest.

iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or apply analysis rules to the values of interest in the response

UI grouping/location		**iTest result type**
Port Traffic	**Basic Traffic Results**	AnalyzerPortResults, GeneratorPortResults
	Diffserv Results	DiffServResults
	Port Average Latency Results	PortAvgLatencyResults
	Overflow Results	OverflowResults
	Port Pair Results	Tx/RxPortPairResults
	CPU Port Results	Tx/RxCpuPortResults
Stream Results	**Traffic Group Results**	Tx/RxTrafficGroupResults
	Filtered Stream Results	FilteredStreamResults
	Detailed Stream Results	TxStreamResults, RxStreamSummaryResults
	Stream Block Results	Tx/RxStreamBlockResults

Port Protocols	**SONET Interface Results**		SonetResults
	POS Interface Results		PppProtocolResults
	ARPND Results		ArpNdResults
	LACP Results		LacpPortResults
	L2TP Results Not supported in iTest 3.1		L2tpPortResults
	PPPoX Results		PPPoEPortResults
	DHCP Results		Dhcpv4PortResults
	DHCPv6 Results		Dhcpv6PortResults
	IGMP Results		IgmpPortResults
	MLD Results		MldPortResults
	EOAM Results		EoamPortResults
Router Protocols	**BGP Results**		BgpRouterResults
	OSPFv2 Results		Ospfv2Results
	OSPFv3 Results		Ospfv3Results
	ISIS Results		IsisRouterResults
	RIP Results		RipRouterResults
	LDP Results		LdpRouterResults
	RSVP Results		RsvpRouterResults
	PIM Results		PimRouterResults
	IGMP Querier Results		IgmpRouterResults
	MLD Querier Results		MldRouterResults
	STP Results		BridgePortResults
	MSTI Results		BridgePortResults
	LDP-RSVP LSP Results	**LDP LSP Results**	LdpLspResults
		RSVP LSP Results	RsvpLspResults
	EOAM Results	**Port Results** Not supported in iTest 3.1	EoamPortResults
		MEG Results Not supported in iTest 3.1	EoamMegResults
		CC Results Not supported in iTest 3.1	EoamContChkLocalResults
		LB Results Not supported in iTest 3.1	EoamLoopbackResults
		LT Results Not supported in iTest 3.1	EoamLinkTraceResults

Host Protocols	**DHCP Results**	**Dhcpv4BlockResults**	
	PPPoX Results	**PPP/PPPoeClientBlockResults, PPP/PPPoeServerBlockResults** Not supported in iTest 3.1	
	DHCPv6PD Results	**Dhcpv6BlockResults**	
	L2TP Results	**L2TPv2BlockResults** Not supported in iTest 3.1	
	IGMP Results	**IgmpHostResults**	
	MLD Results	**MldHostResults**	
	IGMP-MLD Group Results	**IGMP Host-Group Results**	IgmpGroupMembershipResults
		MLD Host-Group Results	MldGroupMembershipResults
	SIP Results Not supported in iTest 3.1		SipUaBlockResults
IPTV	**Test Results**		IptvTestResults
	Port Results		IptvPortResults
	STB Block Results		IptvStbBlockResults
	Viewing Profile Results		IptvViewingProfileResults
	Channel Results		IptvChannelResults

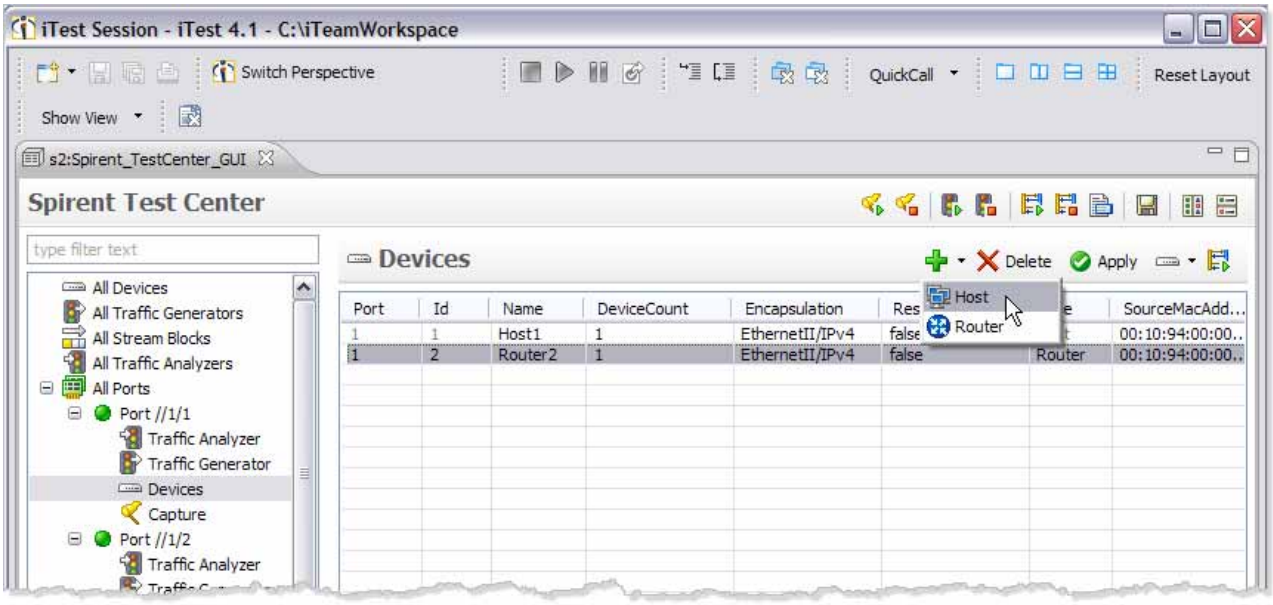
Capture actions and analyze data

The following sections provide overviews for common testing situations. You perform all actions (except getting data) the same way as in TestCenter.

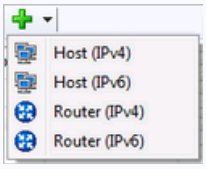
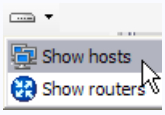
Descriptions for the actions appear in “Spirent TestCenter Command reference” on page 1048.

To add and manage devices (hosts and routers)

When you select **All Devices** or **Devices** in the **Model Objects** tree, the session window displays the list of hosts and routers.



To perform an action on a host, router, or device, click one of the buttons in the toolbar. Descriptions for the actions appear in "Host commands" on page 1058, "Device commands" on page 1059, and "Router commands" on page 1060.

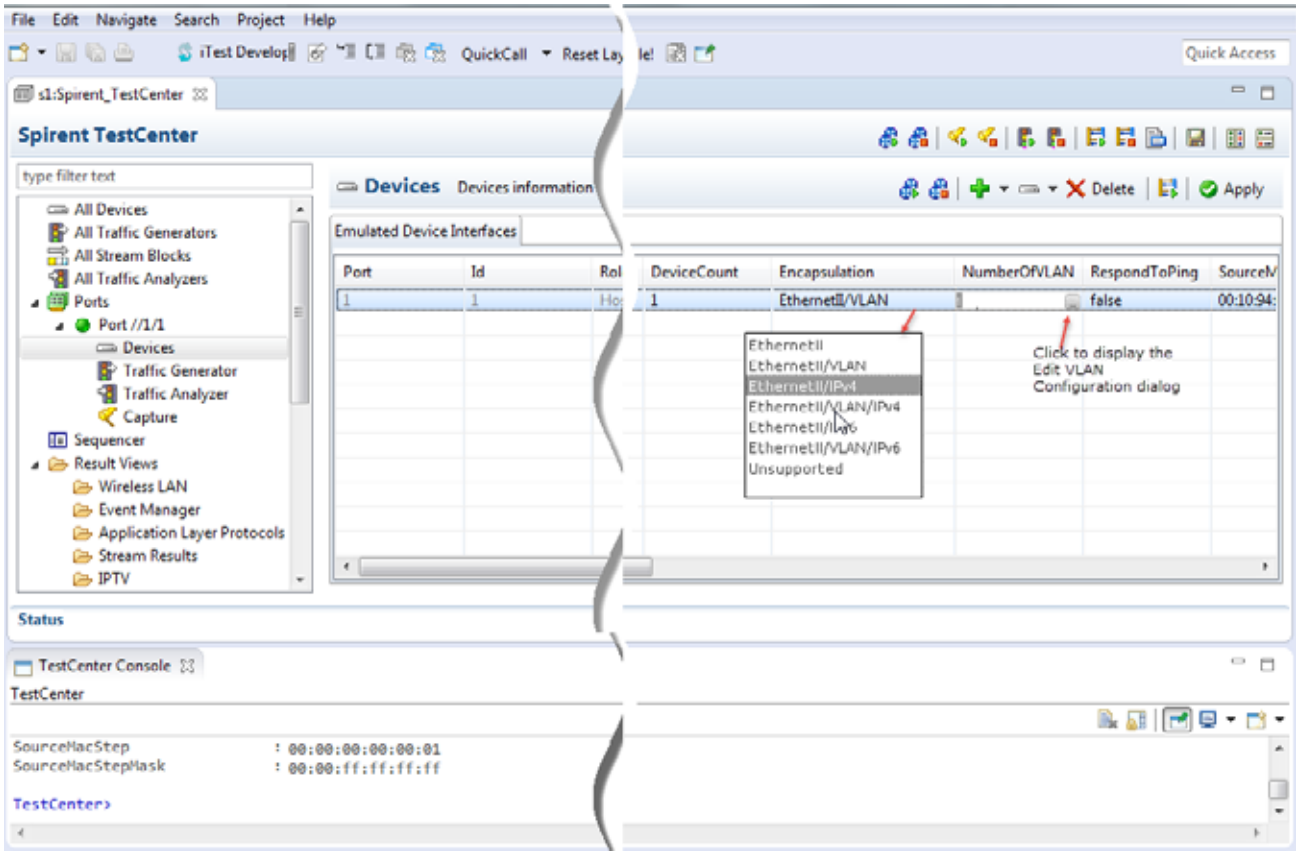
Toolbar button	Captured iTest action
	**addHost** **addRouter**
	**deleteDevice**
	**configureDevice**
	**showHosts** **showRouters**

Configure VLAN on Testcenter GUI

iTest TestCenter session supports configuring VLAN encapsulation for the device. The VLAN configured device can then send and receive packets with another device within the same VLAN ID (for testing quick traffic blasting within VLANs). VLAN encapsulation is supported:

- On devices and bound stream blocks
- In VLAN QinQ
- In REST and TCL-based session types

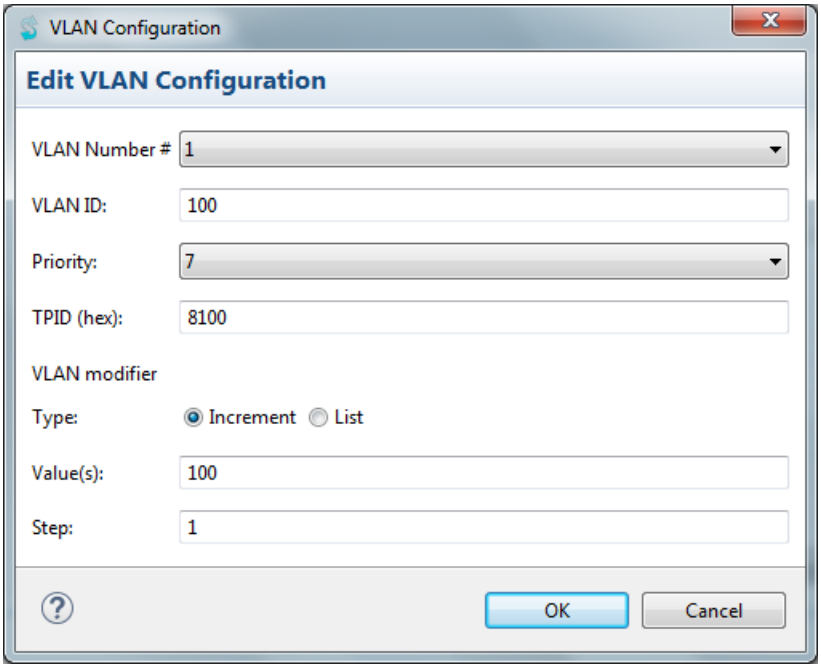
In addition to supporting configuring VLAN encapsulation settings on the iTest TestCenter GUI, iTest supports viewing and configuring VLAN settings on iTest TestCenter console exactly as you do in the TestCenter. See "Configure VLAN on TestCenter Console page" on page 1080 and "Configure VLAN on TestCenter Testcase step" on page 1081. iTest Spirent TestCenter session editor also allows you to view and configure VLAN settings. On the device page, the column, NumberOfVLAN (button) allows you to view and configure VLAN settings.



To configure VLAN encapsulation, select a device in the device table and configure VLAN Encapsulation for the selected device as follows.

- **Encapsulation** column: Select type of VLAN (Ethernet/VLAN, Ethernet/VLAN/IPv4, Ethernet/VLAN/IPv6) from the list.
- **NumberOfVLAN** column: Input number of VLANs to be configured and then click the button to view and configure VLAN settings.

VLAN Encapsulation displays in GUI and console after loading a VLAN configuration in the session. The NumberOfVLAN column displays in the session only when the device is configured with VLAN encapsulation. You can add VLAN encapsulation, configure VLAN encalsulation properties as required.



Field Name	Type	Default Value	Boundary Value
VLANs per Port	Integer	1	[1 ; 4095]
VLAN ID	Integer	-	[0 ; 4095]
Priority	Integer	7	[0 ; 7]
TPID(hex)	String	8100	[0 ; 9] & [A ; F]

VLAN Modifier

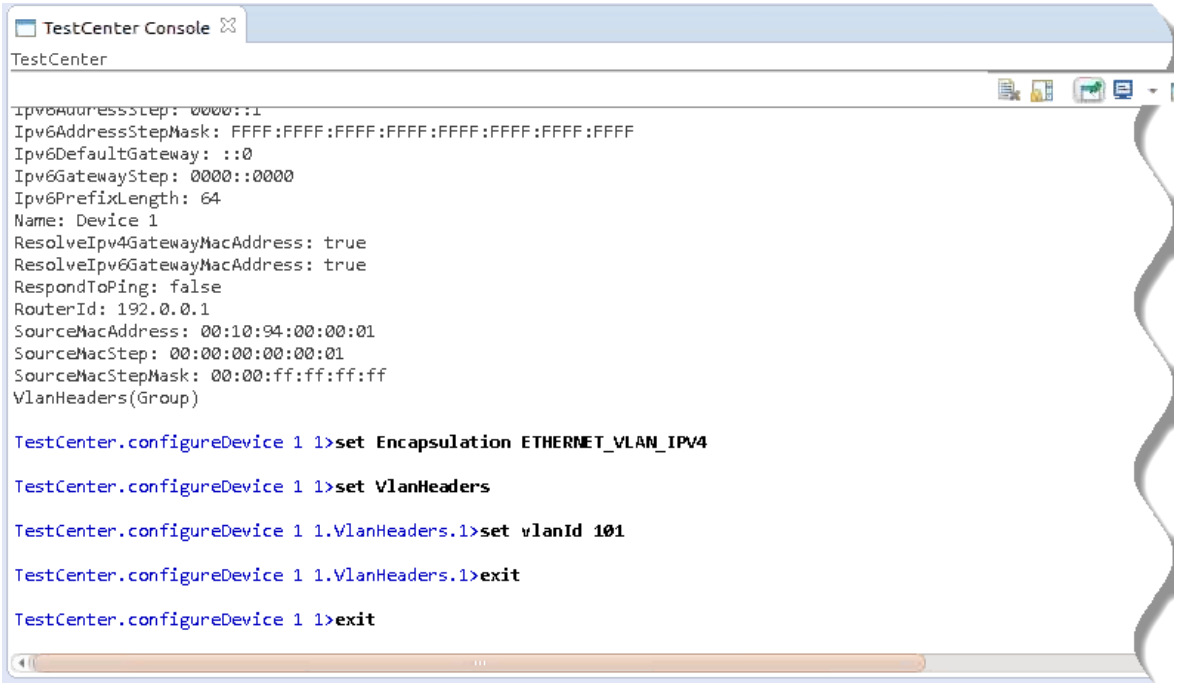
Field Name		Type	Default Value	Boundary Value
Modifies Type: **Increment**	Value	Integer	100	[0 ; 4095]
	Step	Integer	-	
Modifies Type: **List**	Value	Integer	-	
	Step	N/A (Disabled)	8100	

Note These values are not allowed in the above fields.

- Negative numeric (such as: -999)
- Rational numbers (such as: 9.9)
- Alphabet characters (such as: ABCabc)
- Special characters (such as: !@#$%^&*)

Configure VLAN on TestCenter Console page

To add/edit VLAN configuration, load the session file, and change or add VLAN configuration as required. At the TestCenter console, enter **Help** to display the list of parameters/groups available for setting or modifying values, as required.

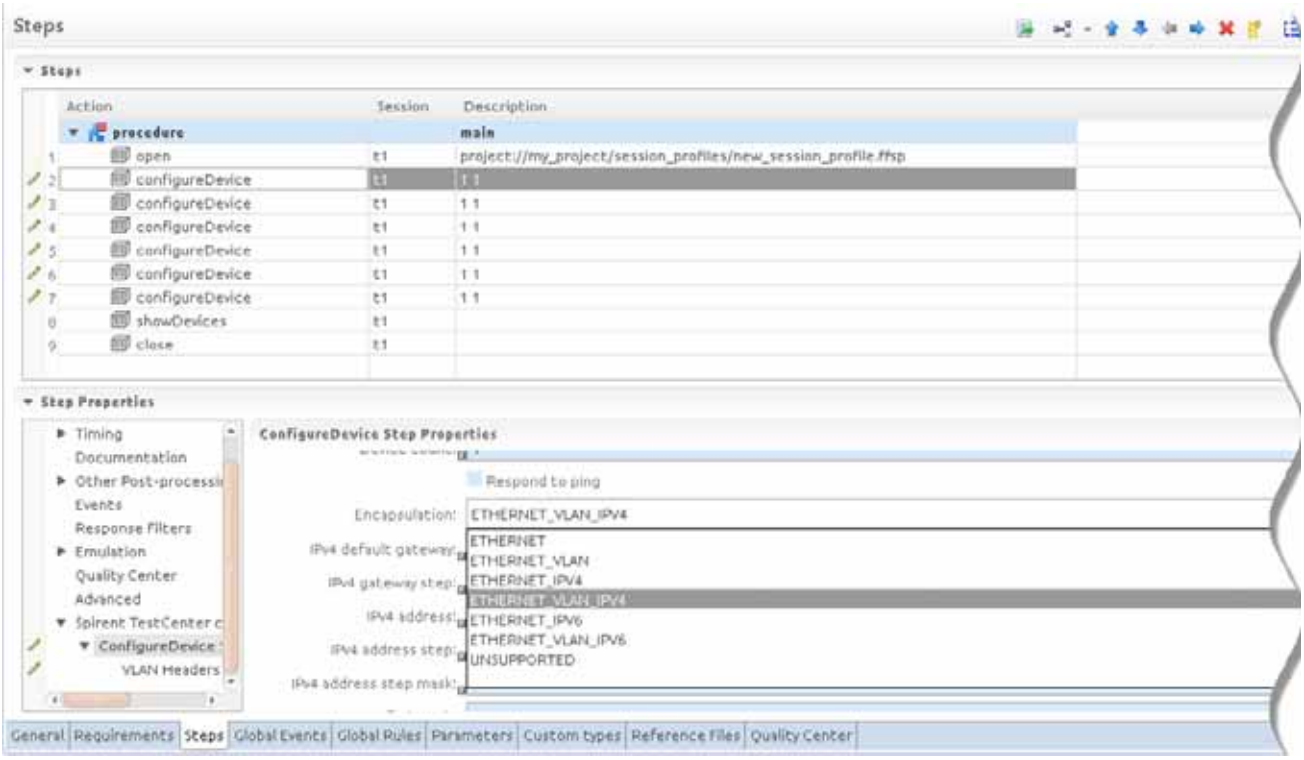


You may change VLAN properties as required and apply the new values to the device.

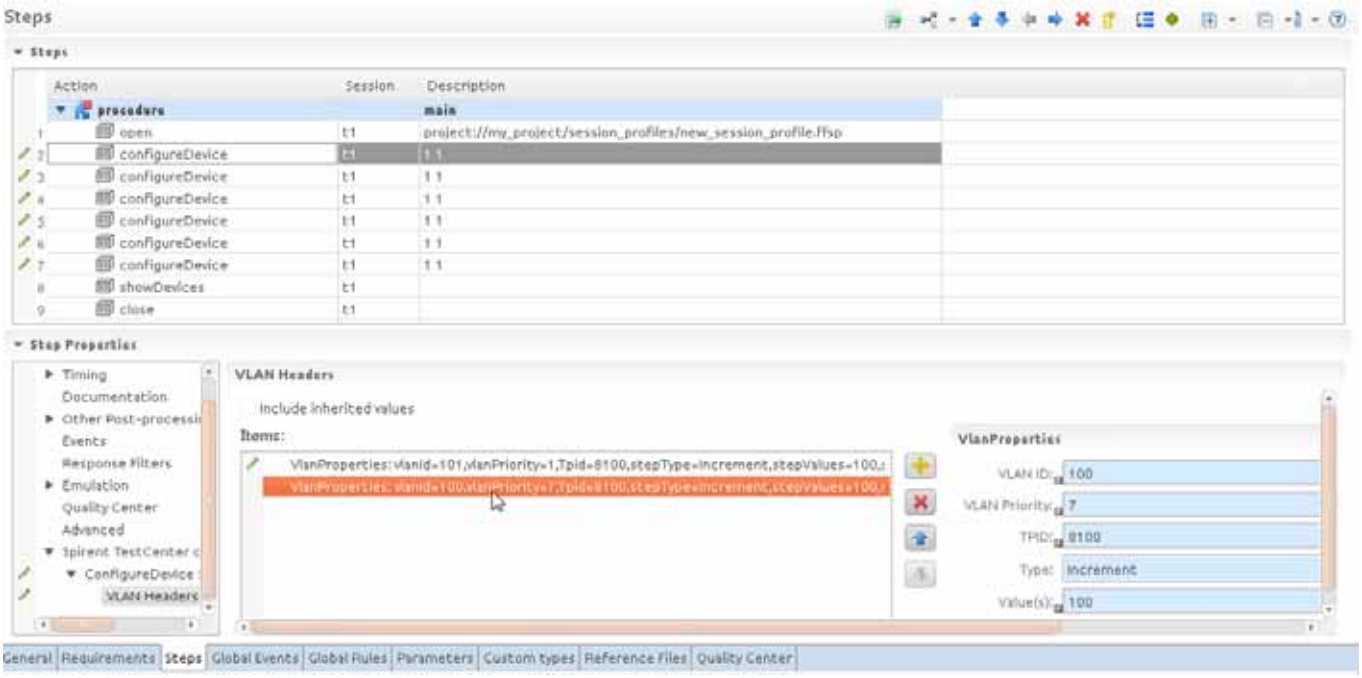
Note You may also use the TestCenter REST Console to set VLAN Encapsulation. See Chapter 54, “Spirent TestCenter REST sessions.”

Configure VLAN on TestCenter Testcase step

Open session type Spirent TestCenter. Select the ConfigureDevices Step properties, add VLAN encapsulation and configure properties as required.



Add VLAN Headers and configure as required..

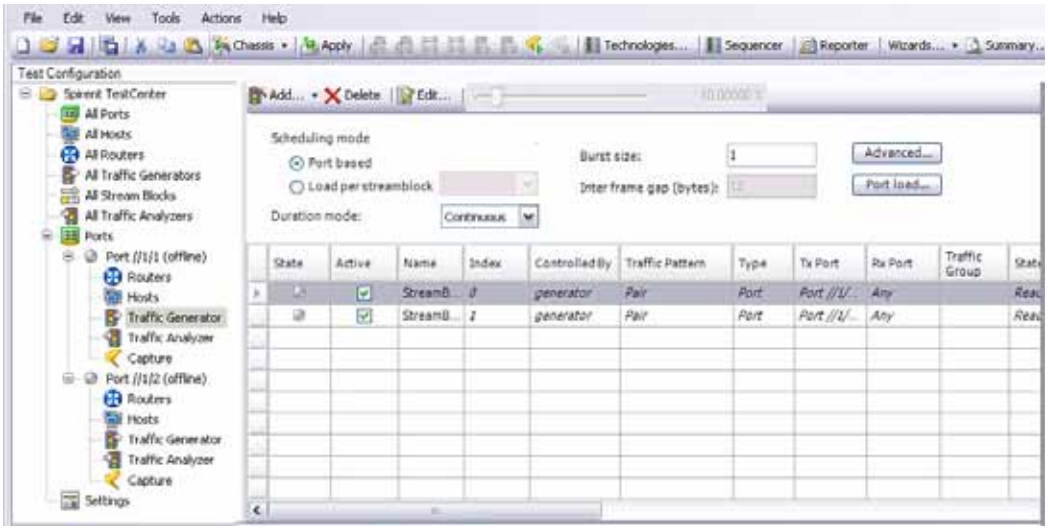


To configure traffic on a port

You create and configure a traffic generator in the same way as on TestCenter. To perform an action on the generator, click one of the buttons in the toolbar.

In addition to supporting 10G, 40G, and 100G traffic and port settings, iTest supports viewing and configuring 10G, 40G, and 100G port settings on iTest TestCenter console exactly as you do in TestCenter. See "Configure Port on Spirent TestCenter Console page" on page 1083 and "iTest Spirent TestCenter Session page" on page 1085. iTest Spirent TestCenter session editor also allows you to view and configure port settings of Ethernet and IEE802.11 wireless traffic port.

Spirent TestCenter page



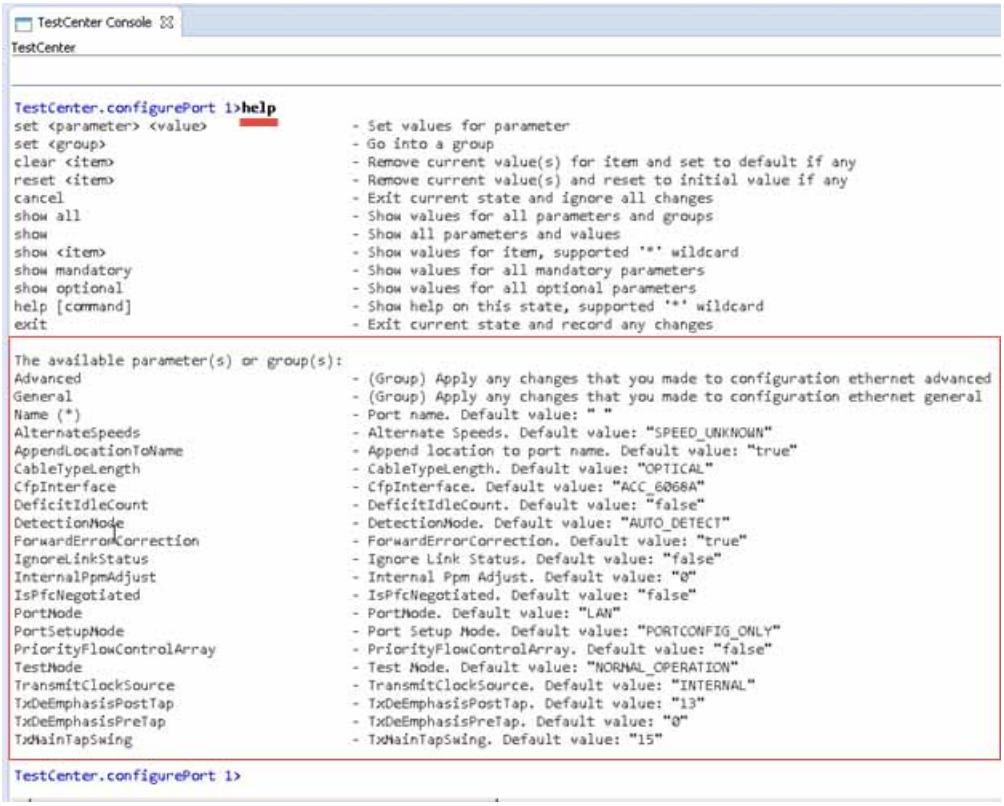
iTest TestCenter page

To perform an action on the generator, click one of the buttons in the toolbar. Descriptions for the actions appear in "Spirent TestCenter Command reference" on page 1048.

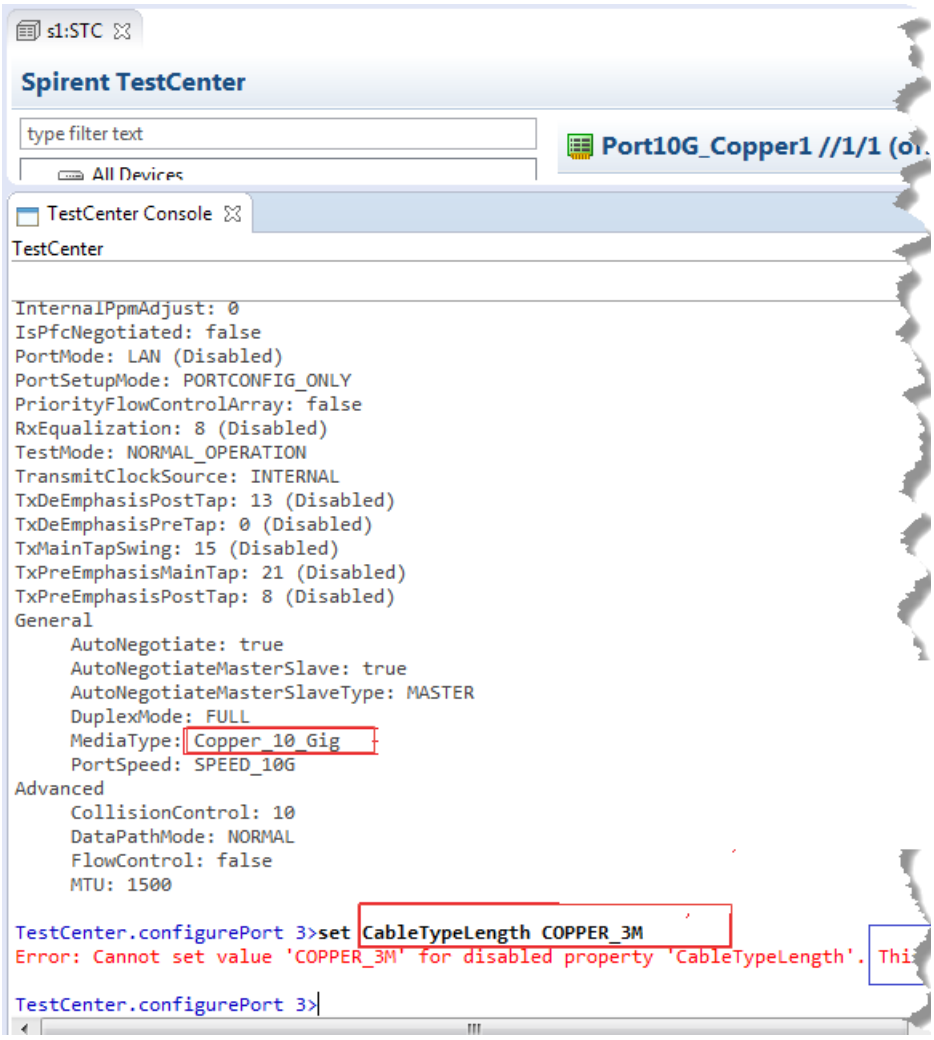
Toolbar button	Captured iTest action
Add Raw Stream Block	**addRawStreamBlock**
Delete	**delete**
Edit...	**edit**
	startCapture
	stopCapture
	startGenerator
	StopGenerator
Apply	**configureGenerator**

Configure Port on Spirent TestCenter Console page

To edit a port configuration file, load file, for example, a10G port configure file and change or add configuration as required. At the TestCenter console, enter **Help** to display the list of parameters/groups available for setting or modifying values, as required.



Show all properties or physical port, to view the properties that may be changed (notice that the example has some properties that has been disabled).

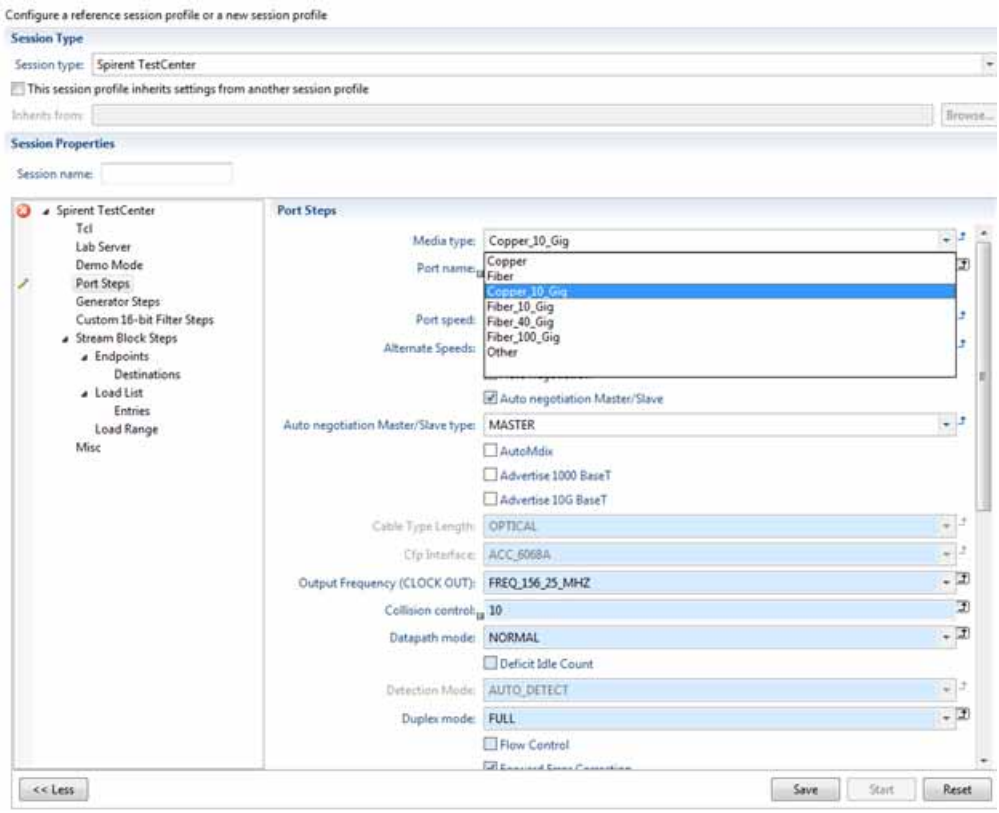


You may change properties as required and apply the new values to the device. If any property has been disabled for the port, your changes will not be applied. That is, changes applicable for 10 Gig Fiber cannot be applied to 10 Gig copper and vice versa. An error displays providing you the reason for failure.

Note For example: Cannot set value 'Copper_3M' for disabled property 'CableLength'. This argument is only enabled when 'Mediatype' argument pattern match 'Fiber_10_Gig\Fiber_40_Gig\Fiber_100_gig'.

iTest Spirent TestCenter Session page

Open session type Spirent TestCenter. Select the media type and appropriate settings as required. The attribute that are not supported for a particular port type are not available for configuration.

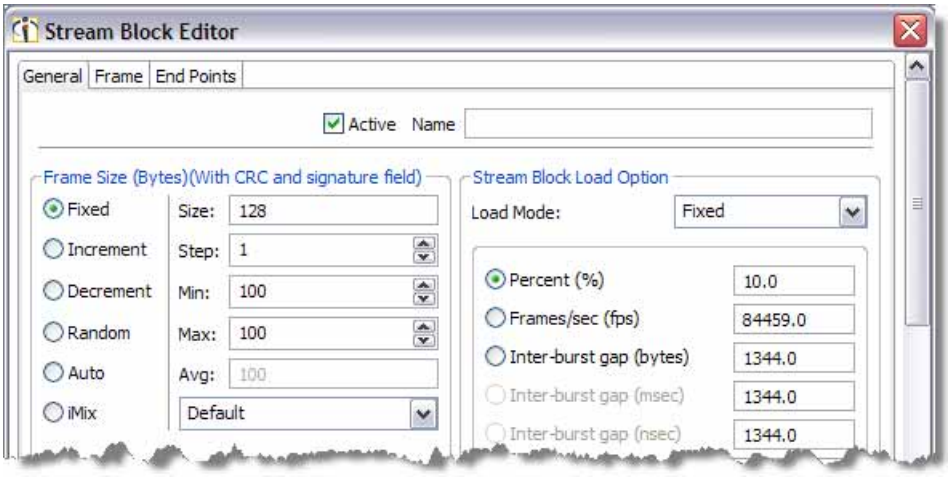


To add a raw stream block

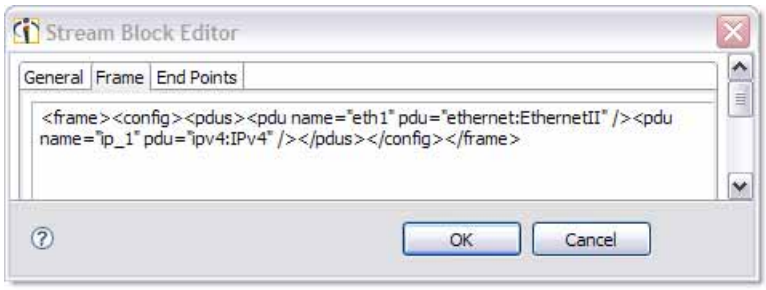
Note iTest sessions can add raw stream blocks and does not add bound stream blocks.

To add a raw stream block

1 Select a generator and click Add . iTest opens a **Stream Block** editor that is very similar to the editor that you are used to using in TestCenter.



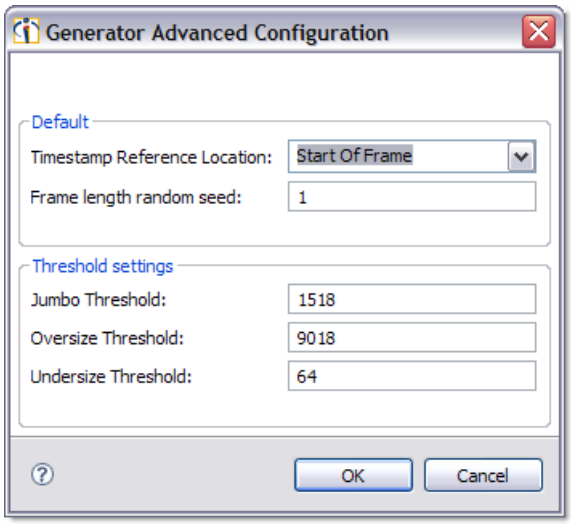
The iTest **Frame** tab differs from the TestCenter **Frame** tab. You specify the hex contents of the entire frame in a text box. You construct the text using other tools.



2 When you have finished configuring the stream block, click **OK**. iTest captures an **addStreamBlock** action and captures all of the settings that you configured in the editor. In the iTest Test Case editor, you can view (and edit) all of the captured property settings in the **Step Properties** section for the step.

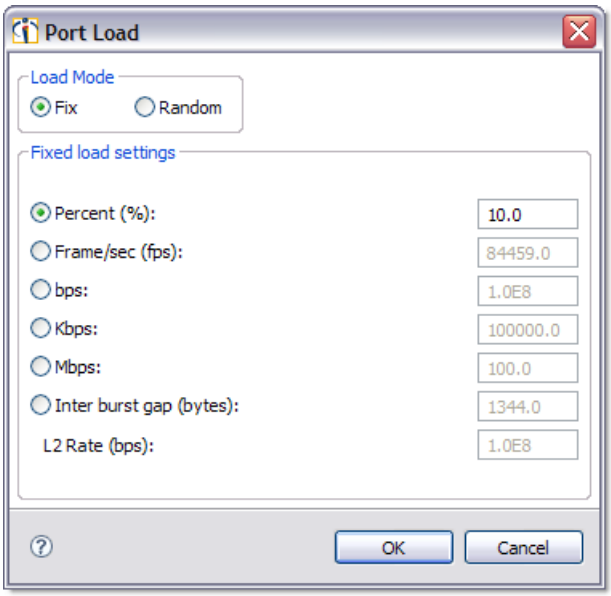
The Generator Advanced Configuration dialog box

Specify settings in the dialog box exactly as you do in TestCenter.



The Port Load dialog box

Specify settings in the dialog box exactly as you do in TestCenter.



To add a traffic filter

Select the traffic analyzer and click . iTest executes an **add16BitFilter** action. The description for the action appears in "Spirent TestCenter Command reference" on page 1048.

To capture on a particular port

You configure capture in a iTest TestCenter session exactly as you do in TestCenter. You also define capture patterns in exactly the same way.

To display results (Statistics nodes)

iTest list results in the same tree that it uses to list devices, in a folder named **Statistics**. When you select an item, the data appears in the **Console** view.

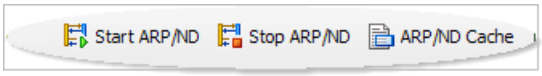
To find items: Use the filter

You can limit the events that appear in the list by typing part or all of an event name into the text box at the top. Event names that match the filter are displayed. The * and ? wildcard characters are supported.

To reset the text so that no filter is applied, delete all text from the text box.

Work with ARP packets

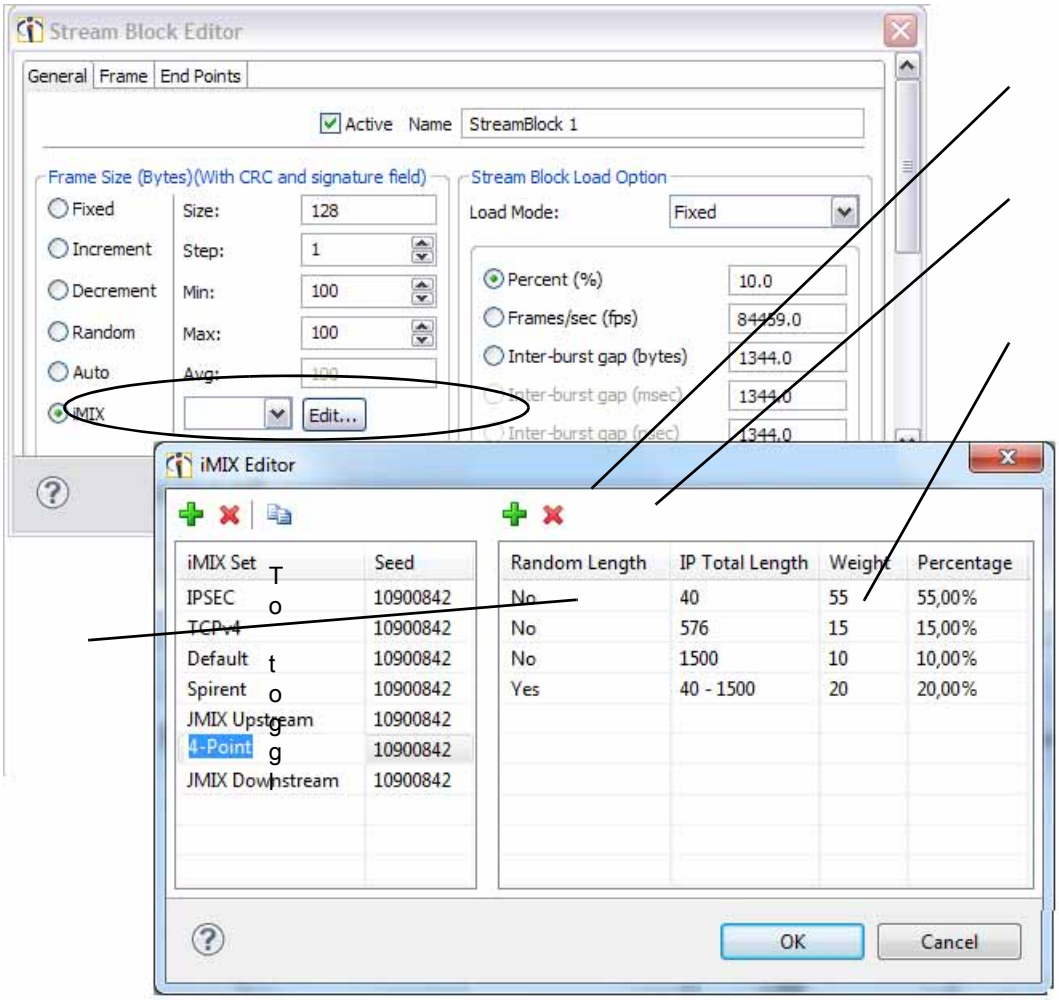
You can generate ARP packets for devices, for specified ports, and for all ports. Commands that start and stop ARP packets return the ARP/ND state (**WAITING**, **SUCCESSFUL**, **CONGESTED**, and so on). In addition, you can view the ARP/ND cache.



See “ARP packet actions” on page 1060.

To edit iMIX settings

To open the iMIX editor, select a stream block and click **Edit**. In the **Stream Block** editor, select **iMIX** and then click **Edit**. See “iMIX commands” on page 1055.



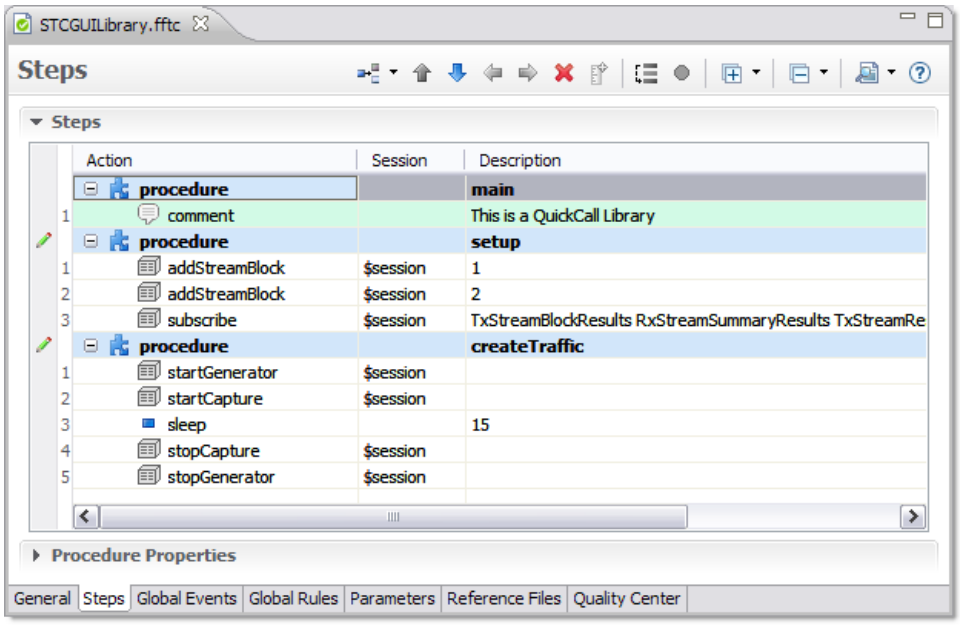
Captured commands:

addFrameLengthDistribution

deleteFrameLength

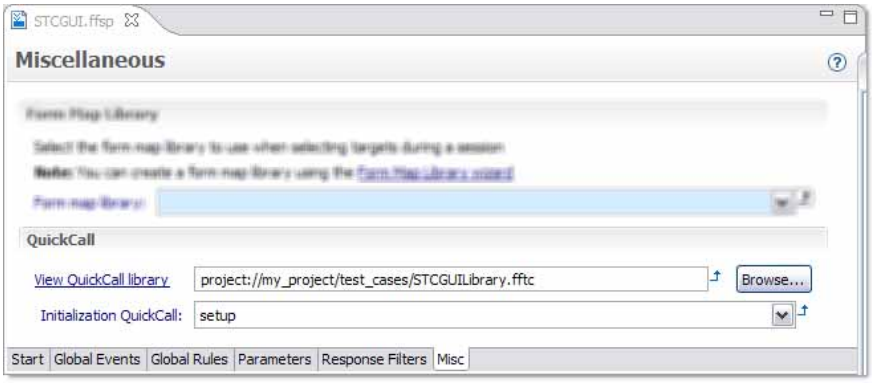
Example QuickCalls for TestCenter tests

We define two QuickCalls in the QuickCall library named **STCGUILibrary**: **setup** and **runTraffic.**



The ‘setup’ QuickCall

The **setup** QuickCall adds two stream blocks and subscribes to several statistics of interest. We’ll specify **setup** as the **Initialization QuickCall** on the **Miscellaneous** page of the session profile so that it executes as the very first step whenever a session starts (either manually or as part of an automated test case):



The ‘runTraffic’ QuickCall

The **runTraffic** QuickCall generates and captures 15 seconds worth of traffic. We can execute it as a single mouse-click during manual testing or as a QuickCall step in a test case.

Spirent TestCenter session profiles

Session profile property settings for Spirent TestCenter sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Spirent TestCenter properties

Chassis IP	Specify the IPv4 address or DNS hostname of the device. **Note** If you do not specify a **Chassis IP** value, then you can use **//chassis/slot/port** notation in the **Ports** property to refer to ports on multiple chassis.
Ports	Specify a single port or list of ports for the session. See "To specify a list of port locations" on page 1050. If no **Configuration file** is specified, then, when the session starts, one port is created for each location in the list. The ports are then connected to and reserved. **Note** iTest assigns ports in the listed order. For example, "1:9,1:8" assigns Port 9 first and "1:8,1:9" assigns port 8 first. If you are loading a configuration file and port order is important, you must specify ports in the same order as in the configuration file. **Note** In addition to supporting 10G, 40G, and 100G traffic and port settings, iTest supports you to view and configure 10G, 40G, 100G port settings in TestCenter.console. iTest also supports viewing and displaying of IEEE802.11 port type.

Configuration file	Optional: Specify the configuration file (either XML or tcc format file) to use to configure the device when the session starts. The path is limited to 256 characters. You can generate a configuration file using the TestCenter **configuration save** command. When the session starts: • If no port locations are specified by the **Ports** property, ports in the configuration file with valid locations are connected to and reserved. Offline or inactive ports (**Active** attribute false) are ignored. • If the **Ports** property specifies the same number of port locations as in the configuration file, the ports in the file are mapped to the listed locations in order. • If the **Ports** property specifies fewer port locations than the file, then as many ports as possible are mapped to the listed locations. The mapping begins with the first port in the file and continues until the list is exhausted. All mapped ports are activated and then connected to and reserved. Any ports remaining in the file that have valid addresses and are active are also connected to and reserved. • If the **Ports** property specifies more port locations than the file, then all ports in the file are mapped to listed locations. Mapping begins with the first listed location and continues until all ports in the file are exhausted. All mapped ports are activated. Any locations remaining in the list specified for the **Ports** property are used to create ports. All ports are connected to and reserved.
Force taking port ownership	Upon connecting, take ownership so no other user can submit commands.

Tcl

We recommend that you use the default values for the following properties.

Use global Tcl interpretor during execution	Select to indicate that the Tcl Interpreter used during execution is the one specified in **Preferences > Spirent > Session Types > Spirent TestCenter > Use specified Tcl Interpreter**
Path to Tcl interpretor	Enter the path of the Tcl interpreter to be used during execution, if you do not wish to use the Global Tcl Intepreter. Default: Blank
Spirent Test Center Tcl API directory	Optional. Specify the path to the folder that holds the Tcl API to use in place to the default system Tcl API. Default: blank

LabServer

The following properties let yo u specify whether and how to use Spirent TestCenter Lab Server for iTest sessions.

Use LabServer	Check the box to use the LabServer option and run Spirent TestCenter on a server, as specified by the following settings.
Server IP/hostname	Specify the IP address or hostname of the host that is running Lab Server

Create new session on connect	If checked, use the credentials specified by the following properties to start a new LabServer session. If unchecked, use the currently running LabServer session for the Spirent iTest session. Works in conjunction with the **Terminate session on disconnect** setting.
Session name	Name to use to identify the session when starting a new LabServer session.
Owner ID	Owner ID to use when starting a new LabServer session.
Terminate session on disconnect	If checked, then end the LabServer session when Spirent iTest disconnects from LabServer. If unchecked, then do not end the LabServer session. Works in conjunction with the **Create new session on connect** setting.

Demo Mode

Run session in demo mode	If unchecked (default), TestCenter sessions on iTest run normally, sending commands to the device and collecting data. If checked, iTest runs the session against prepared data, enabling you to view results as if it were being returned by a device. The intent is that you can run a session to learn more about how TestCenter sessions operate without having to install or run any Spirent software. **Important** If you specify demo mode, then you must specify a value for each of the required properties in the **Spirent TestCenter** group, as described in "Spirent TestCenter properties" on page 1091. (Any settings will do — iTest does not use the values. The values are required only to satisfy the iTest validation checks.) **Supported statistics** Only the following statistics are supported in Demo mode. Trying to demo any other Statistics will result in an exception. AnalyzerPortResults GeneratorPortResults RxPortPairResults TxPortPairResults RxStreamSummaryResults RxStreamBlockResults TxStreamBlockResults TxTrafficGroupResults RxTrafficGroupResults TxStreamResults Default: unchecked

Port Steps, Generator Steps, Custom 16-bit Filter Steps, Stream Block Steps

Note The following sets of session profile settings correspond directly with TestCenter configuration settings. The settings are fully described in Spirent documentation and are not described here. Notice that all of the settings also appear as step properties for any Spirent TestCenter step in a test case.

Port Steps

Generator Steps

Custom 16 bit Filter Steps

Stream Block Steps

Misc

Display slot:port format in responses	When iTest returns responses to commands, it can display port references either in **slot:port** format (for example, **2:3**) or in sequential **portIndex** format (just the port identifier, for example, **5**). Check the box to display in **slot:port** format. Uncheck to display in **portIndex** format. Default: unchecked
Use slot:port format in captured steps	When iTest captures responses to commands, it can create test case steps that use port references in either **slot:port** format (for example, **2:3**) or in sequential **portIndex** format (just the port identifier, for example, **5**). Check the box to create command arguments in **slot:port** format. Uncheck to use **portIndex** format. Default: unchecked
Use zero-based stream block indices	Check the box to begin counting stream blocks at 0 (zero). That is, the first stream block is 0. If unchecked, then the first stream block is 1.

CHAPTER 76

Spirent TestCenter NTAF sessions (Obsolete and Deprecated)

Important

The Spirent TestCenter NTAF sessions are ***obsolete and is no longer supported***.

This chapter is provided only to support existing implementations.

Overview: Capturing a Spirent TestCenter NTAF test case in Spirent iTest

1. In iTest, when you start a Spirent TestCenter NTAF session, iTest launches the Spirent TestCenter user interface running on the Spirent TestCenter device.
2. Now you interact with Spirent TestCenter in the normal way. For example, you might reserve ports and perform testing and then close the test session. Ports are released and the configuration is saved.

   While you perform the actions in Spirent TestCenter, iTest captures your actions. In addition, iTest captures the responses returned by Spirent TestCenter for each action.
3. When you are finished working in Spirent TestCenter, you return to iTest and close the Spirent TestCenter NTAF session window. iTest disconnects the Spirent TestCenter session and closes the user interface.
4. You can now save the captured steps and responses as an iTest test case. As needed, you can modify and update the test case (for example, modify the **InstallFirmware** step by causing it to load a different version or replace the version number with a variable whose value is set at runtime).

   You can execute the test case at any time. When the iTest test case executes, iTest starts a Spirent TestCenter session and executes the steps exactly as you performed them during your manual session.

Responses are mapped

iTest does not use the Spirent TestCenter protocol to collect or analyze the traffic data—other steps in your test case can do that.

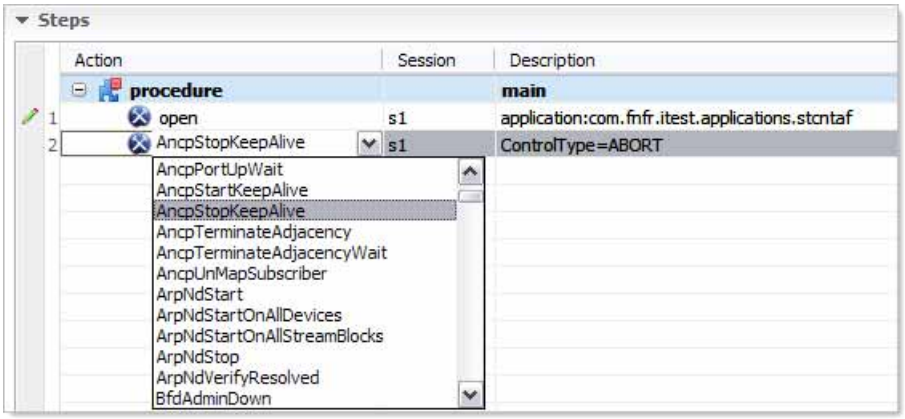
iTest saves all responses to Spirent TestCenter NTAF steps as structured data and auto-maps the responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries.

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Spirent TestCenter NTAF action reference

The name of each iTest action is the same as the name of the Spirent TestCenter Tcl API function that it executes. See the *Spirent TestCenter Automation Programmer's Guide* for details on operation and usage.



If the Spirent TestCenter Tcl command associated with the iTest step/action requires arguments, then, for the selected step in the Test Case editor, you can view and set the argument values in the **Step Properties**.

Each iTest action returns the same values as the corresponding Spirent TestCenter command.

Note For **open** steps in test cases, the **filepath** parameter is required (it specifies the path of the configuration file). The **portLocation** parameter is optional (enables you to set the port at runtime).

To add a Spirent TestCenter NTAF step to a test case

You can create a test case by typing or selecting actions to add steps in the Test Case editor, but the easiest way to create a test case is either the **direct-to-test capture** or **saving a captured session as a procedure** methods, as described in Chapter 7, "Test Cases".

Starting a session with Spirent TestCenter NTAF

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Step 1 Log in to the NTAF server

Follow the instructions in Chapter 70, "Working with NTAF sessions in Spirent iTest".

Step 2 Define a Spirent TestCenter NTAF session

Ensure that the Spirent TestCenter NTAF session profile or device is properly configured. See “Session profile property settings for Spirent TestCenter NTAF sessions” on page 1432.

Step 3 Start a session with Spirent TestCenter

Use either of the following methods to start a Spirent TestCenter NTAF session in iTest.

- In the Topology view, select and start the session. This is the typical method for manual testing.
- In the **Session Profile** editor, open the session profile and click **Start**.

Note To allow maximum flexibility and portability, iTest uses the NTAF standard to communicate with the Spirent TestCenter session. As a result, you often see the term “NTAF session” associated with Spirent TestCenter sessions in iTest.

Setting preferences for NTAF sessions

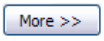
All NTAF preference settings are used by default whenever a user attempts to connect iTest or the NTAF Proxy service to the NTAF server. The settings are applied to all NTAF-compliant providers — not only to the particular type of NTAF session that you are currently working on.

See “Setting preferences for NTAF sessions” on page 1311 in Chapter 70, “Working with NTAF sessions in Spirent iTest”.

Session profile property settings for Spirent TestCenter NTAF sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Harness

An NTAF harness defines the action set for a iTest session with an NTAF provider (like Spirent Test Center or Landslide).

As new provider versions are released, new versions of the associated NTAF harness may also be released to implement the latest commands and features. The harness version number is typically identical to the provider version number.

Harness URI	Harness version to use for submitting actions to a session. **Note** New harness versions are designed to operate properly for test cases built using older harnesses, as long as the newer version supports the old harness or the subharnesses used in the test case. The **Action** cell in the Test Case editor list all actions supported by the harness and its subharnesses. If an action from an older version is no longer supported by the new version, then an error appears for the affected step in the test case. Click the error icon to view a description. Typically, you can select the new version of the action to correct the error. The **Action** cell in the Test Case editor lists actions by their subharness followed by the action name. Old test cases that contain actions specified only by the action name display an error icon by the action. You must select the correct action in list by looking for the correct subharness/action. Once you select the correct subharness/action, the properties are auto-populated.

Important If you change the harness in a session profile, then review all property settings to ensure that the correct settings are used (changed values are highlighted in yellow).

In addition, if you change the harness version in session profile A, then all property settings in session profile B that inherit from A are reset to inherit by default, including values that had been manually specified in B to override the inherited setting. This means that you must edit B after the change to ensure that the override takes place.

For example, for sessions that inherit property settings from another session profile, you cannot change the inheriting harness version. To be able to change the harness version, you must create a session profile that does not inherit property settings.

Spirent TestCenter

NTAF node	Specify the URL or hostname of the XMPP server (the “NTAF server”)
NTAF provider	Specify the version of the Spirent TestCenter NTAF provider to use.
Config file	Specify the filename for the configuration file. If you do not specify a path, then the configuration file is saved to **C:/myConfig.xml** **Note** Only XML configuration files are supported.

Spirent TestCenter > Port Location

portLocation	Hardware port or ports to map to the config file (that is, during execution, the specified ports are used instead of the ports in the file). During capture, the values are not used; instead, the values in the config file are used. **Syntax** Each value specifies one port. *//chassisIPAddress/slotNumber/portNumber* For example, **//10.61.36.100/2/3** refers to port 3 of the card in slot 2 of the chassis at 10.61.36.100. **Note** To specify multiple ports, add each port individually — click the **Add** button for each port. Default: [none] (No ports are mapped)

CHAPTER 54

Spirent TestCenter REST sessions

Spirent TestCenter REST session window

The session window for Spirent TestCenter sessions in iTest has been designed to closely resemble Spirent TestCenter. As a result, you can capture TestCenter session steps using iTest without having to learn a new interface or new command names. You interact with the iTest session almost exactly like you interact with TestCenter.

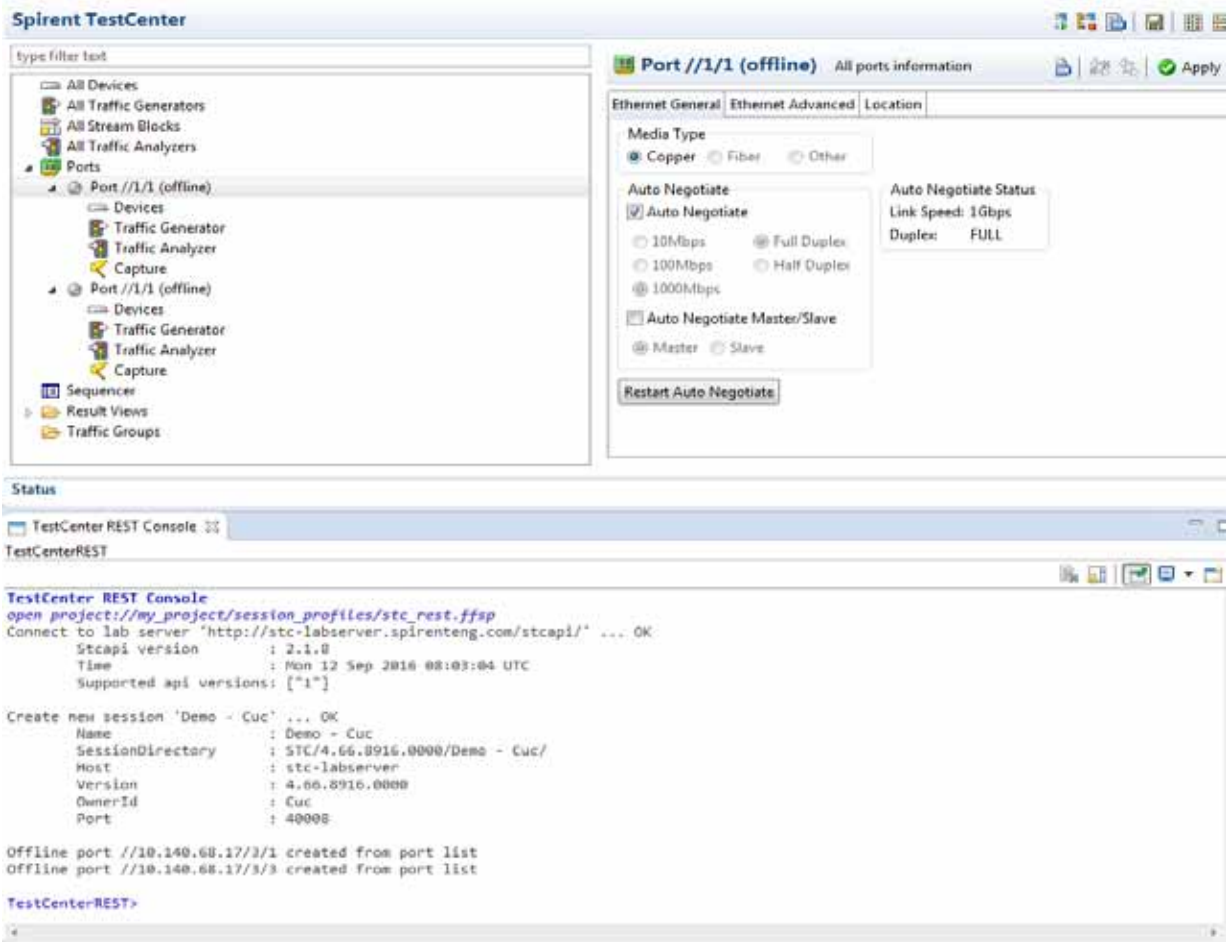
iTest integrates with Spirent TestCenter (STC) and provides REST API to ensure that the automation functionality is readily available to a wide variety of clients using both script and GUI. This integration also eliminates the requirement of an STC installation on the client system. This allows you to use existing tools available to create STC automation clients that can run on any platform.

- The iTest REST API session communicates with STC chassis via STC Lab server or the STCWeb RESTful endpoint using the REST API. When using STCWeb RESTful endpoint, you may automate sessions via REST by installing the STC application software on your desktop instead of creating an STC lab server VM.

  Important To use the STCWeb application, the Spirent TestCenter application must be installed and running on your local PC/Workstation. See the *iTest Installation Guide* for instruction on installing and setting up the Spirent TestCenter application on your local PC/Workstation.

- The iTest REST session can run on multiple platform without requiring the STC libraries.

Because all actions and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent TestCenter.

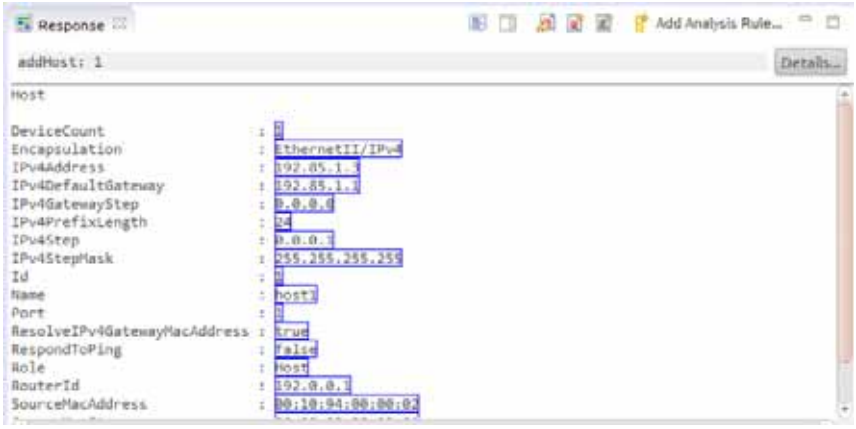


- Any action that you perform in the iTest session is forwarded to Spirent TestCenter running on the device. TestCenter performs the action and returns its normal text response. You can view the response in the **Results** section on the iTest window, just like you do in TestCenter.
- iTest captures all of the actions that you perform in a TestCenter session and all of the responses returned by TestCenter.

Responses are mapped

iTest does not use the TestCenter protocol to collect or analyze the traffic data—other steps in your test case can do that.

iTest saves all responses to TestCenter steps as structured data and auto-maps the responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. Here's the auto-mapped response to an **addHOST** step in an iTest TestCenter REST session:



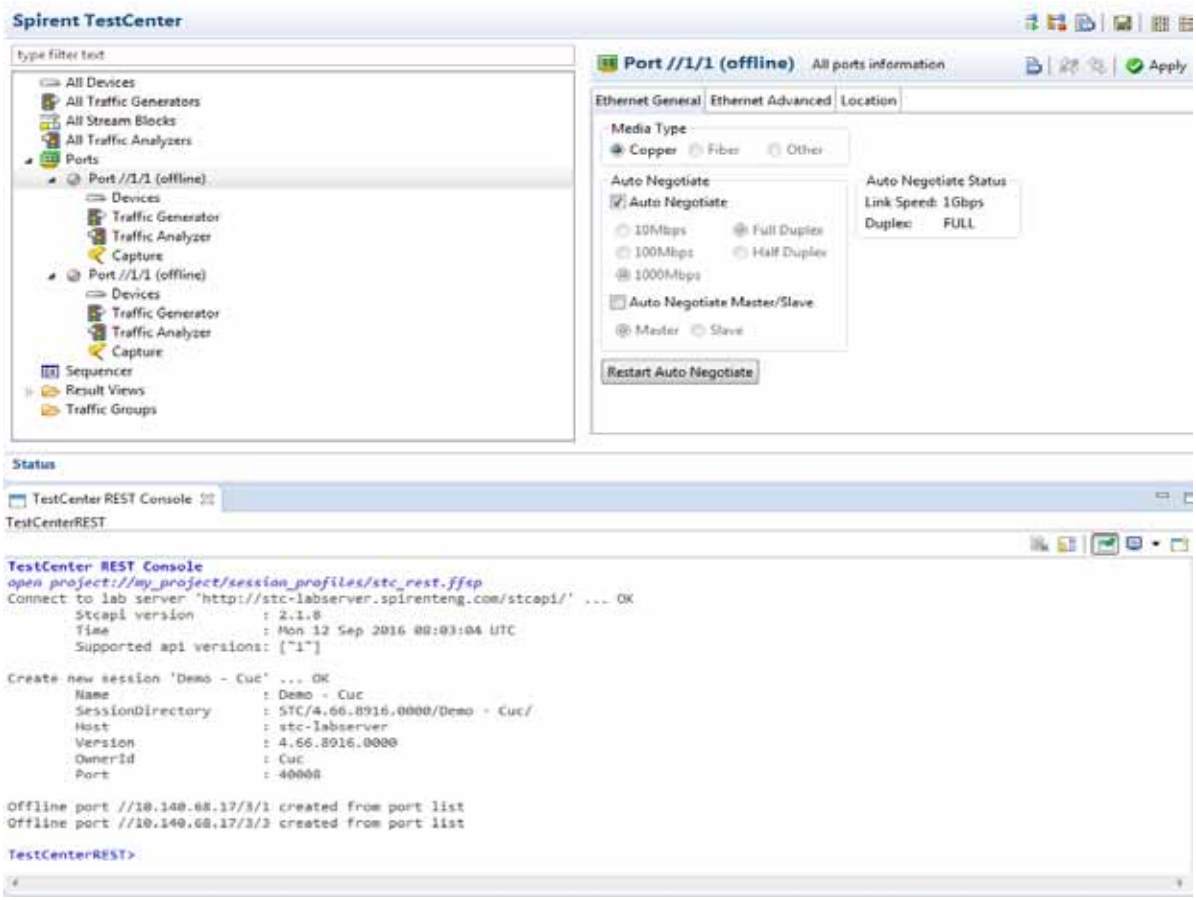
To create a test case that includes Spirent TestCenter REST sessions

You typically save captured manual steps as a test case. Follow these steps:

1 Ensure that the TestCenter REST session profile or device is properly configured. See "Session profile property settings for Spirent TestCenter REST sessions" on page 1106.

2 Click Start Capture to begin the direct-to-test process of saving the interactive session as a test case.

3 Start the TestCenter REST session and perform the test as needed. You work in the iTest TestCenter REST session the same way you work in TestCenter. When you interact with a TestCenter component, iTest performs a TestCenter action and captures both the action and the response from TestCenter. For example:



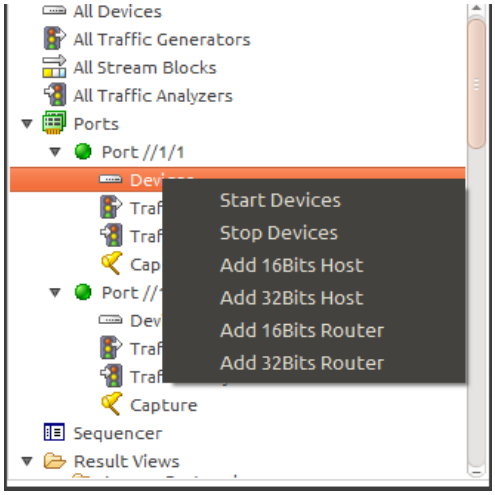
You first select Port 1. When you click Auto Negotiate, iTest opens the **Auto Negotiate Editor** to view or create session (just like TestCenter). Then, when you click **OK** in the editor:

iTest submits command to TestCenter REST on the Spirent device (add a stream block on the generator's port 1).

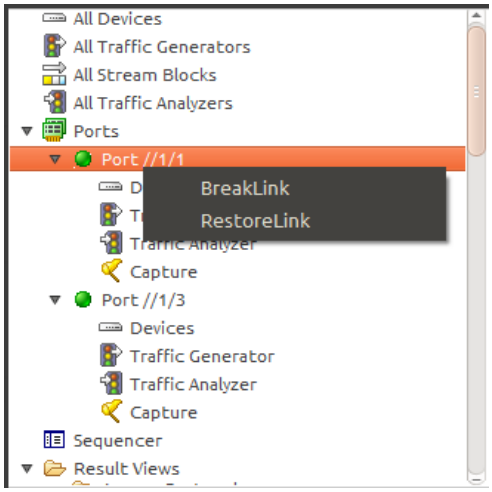
In the TestCenter REST **Console** window, iTest displays the command you entered. The response from TestCenter includes settings that were implemented on the Spirent device.

Note While not all TestCenter commands are available using buttons or other controls on the page, you can perform any TestCenter REST command by entering it on the iTest Console view (as described in "Spirent TestCenter Command reference" on page 1100).

You may also select an object, right-click (for e.g., on Device) to view the properties and perform a required function.



In addition, you may select an object (e.g., Port 1) and right-click, to view the object handlers and their properties, and perform the required action.



4 When you finish, click Finish Capture to save the captured steps into a test case. (The **Add Test Case** wizard opens and help you through the process. For details on saving captured steps to test cases, see "Saving interactive steps as steps in a test case: The 'Add to iTest Test Case' wizard" on page 98.)

5 The Test Case editor opens the new test case. See "To create a test case that includes Spirent TestCenter sessions" on page 1045 for more details.

6 Continue editing the test case by adding analysis rules and flow-control steps as needed.

- For information on working with test cases, see Chapter 7, "Test Cases".
- For information on using the Test Case editor, see Chapter 8, "Test Case Editor".

Additional tasks/views

See "REST API Commands" on page 1101.

On the TestCenter REST Console, You may pick a handler (the object handle is the return value from the function, e.g., create function) by using the analysis Rule. Use the **showHandler** command in capture view.

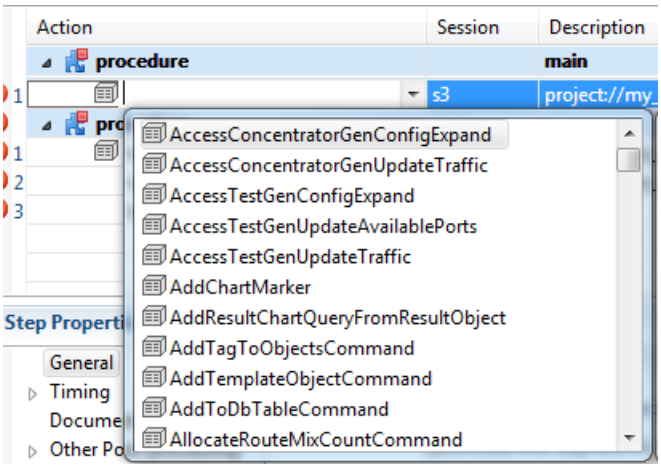
Spirent TestCenter Command reference

This topic describes all TestCenter commands in a session with iTest. The actions are captured during interactive STC sessions and you can add actions as described in "To add a TestCenter REST step to a test case" on page 1101.

iTest TestCenter REST sessions support all commands that are generated by UI elements, **eval** and **source** commands, and all TestCenter **sequencer** commands. See the sections listed below in Chapter 53, "Spirent TestCenter sessions".

To add a TestCenter REST step to a test case

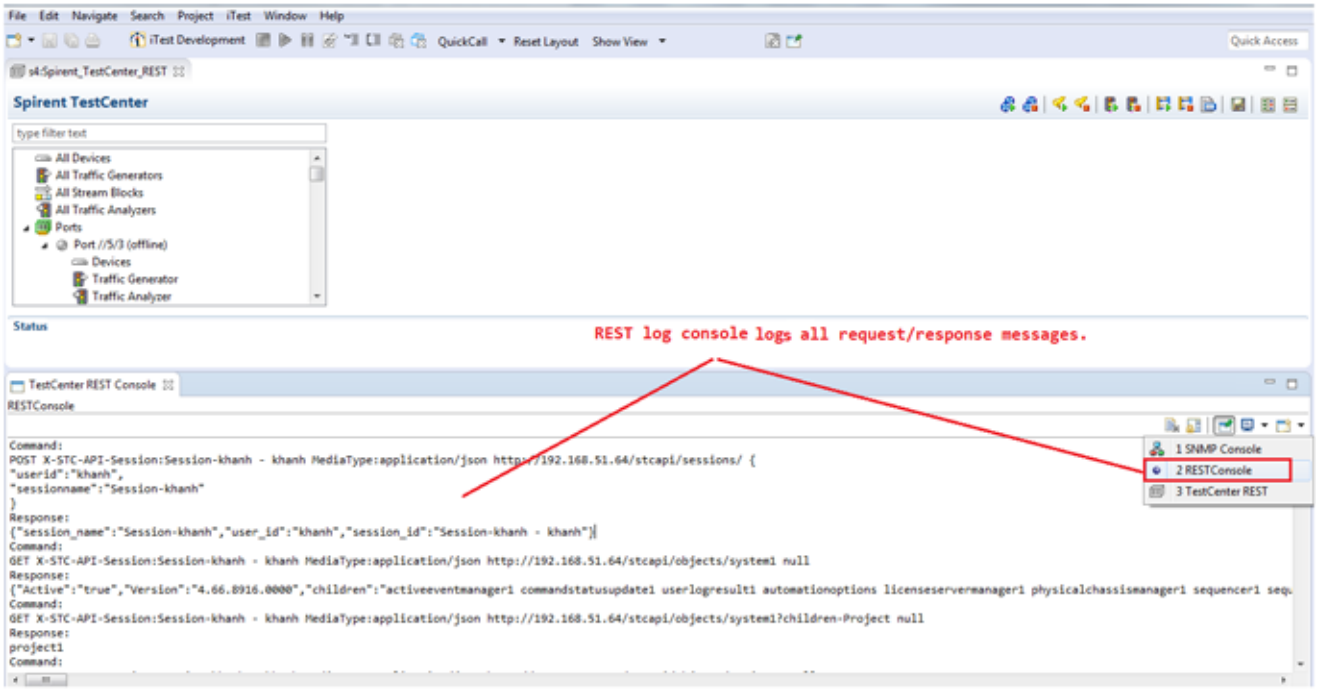
For a step in a TestCenter REST session, click in the **Action** cell (**ctrl + Space**) and select the action from the drop-down list.



See “Spirent TestCenter REST session properties” on page 1107. See also “To add a TestCenter step to a test case” on page 1049.

REST API Commands

To access the REST API commands for TestCenter, open in the iTest TestCenter Console view, type REST Console to access the TestCenter REST console. The example below shows Raw requests to Server and Raw responses from Server



iTest Spirent TestCenter Sessions and REST API

The TestCenter session type incorporates the following commands from the TestCenter REST API integration.

Category	STC Session Commands	REST API
Analyzers	getAnalyzers startAnalyzer stopAnalyzer	GET http://domain:port/stcapi/objects/&portHandle?children-analyzer PUT http://domain:port/stcapi/v1/perform/analyzerStart/ " analyzerList:analyzer1" PUT http://domain:port/stcapi/v1/perform/analyzerStop/ " analyzerList:analyzer1"
Captures	startCapture stopCapture	PUT http://domain:port/stcapi/perform/captureStart "handle:capture1" PUT http://domain:port/stcapi/perform/captureStop "handle:capture1"
Devices	addHost/Router deleteHost/Router showHosts/Routers/Devices configureHost/Router/Device	POST http://domain:port/stcapi/objects/ "object_type:Host/Router, under:project1" DELETE http://domain:port/stcapi/objects/&deviceHandle GET http://domain:port/stcapi/objects/&parenthandle?children-host/router PUT http://domain:port/stcapi/objects/&deviceHandle "propertyname:value"
Ethernet Card	ConfigureEthernet100GigFiber ConfigureEthernet10GigCopper ConfigureEthernet10GigFiber ConfigureEthernet40GigFiber ConfigureEthernetBridgeLink ConfigureEthernetCopper ConfigureEthernetFiber ConfigureEthernetWan ShowEthernet100GigFiber ShowEthernet10GigCopper ShowEthernet10GigFiber ShowEthernet40GigFiber ShowEthernetBridgeLink ShowEthernetCopper ShowEthernetFiber ShowEthernetWan	--N/A--
Port	showPorts addPort deletePort breakLink restoreLink reservePort releasePort mapPort configurePort	GET objects/project1?children-port POST http://domain:port/stcapi/perform/ReservePortCommand/ "object_type:port,under:project1" DELETE objects/&portHandle PUT http://domain:port/stcapi/perform/ReservePortCommand/ "location:&portLocation" PUT http://domain:port/stcapi/perform/ReleasePortCommand "location:&portLocation" POST http://domain:port/stcapi/perform "command:setupPortMappings" PUT http://domain:port/stcapi/objects/&portHandle "body"

Category	STC Session Commands	REST API
Sessions	getSessions getSessionInfo createSesson deleteSession	GET http://domain:port/stcapi/sessions/ GET http://domain:port/stcapi/sessions/&sessionID POST http://domain:port/stcapi/sessions "userid:&userID,sessionname:&userName" DELETE http://domain:port/stcapi/sessions/&sessionID
Chassis	connectChassis disconnectChassis	POST http://domain:port/stcapi/connections/ "action:connect, &chassis:&chassis" DELETEhttp://domain:port/stcapi/connections/&chassis/
Apply	apply	PUT http://domain:port/stcapi/apply
StreamBlocks	showStreamBlocks addStreamBock configureStreamBlock deleteStreamBlock	GET http://domain:port/stcapi/objects/&streamblockHandle POST http://domain:port/stcapi/objects/ "object_type:treamblock, under:port1"
Generators	getPackageGenerator startGenerator stopGenerator	GET http://domain:port/stcapi/objects/&portHandle?children-generator PUT http://domain:port/stcapi/v1/perform/generatorStart/ "generatorList:generator1" PUT http://domain:port/stcapi/v1/perform/generatorStop/ "generatorList:generator1"
Results	subscribeResults showResults saveResults unsubscribeResults	PUT http://domain:port/stcapi/v1/perform/ResultsSubscribeCommand/ "parent:project1, resultParent:port1, configType:generator, resultType:generatorPortResults"
Configure	loadConfigurationFile resetConfig saveConfigurationFile listFiles	PUT /perform/resetConfig "config:system1" PUT /files/&file_name --binary data --.content-type: application/octet-stream, content-disposition: attachment; filename=myconfig.tcc GET /files/&file_name Accept: application/octet-stream GET /files/ Accept: application/json
bll log	getBllLog	GET http://domain:port/stcapi/files/bll.log/
Help	help	GET /help/ command GET /help/&object_type GET /help/&handle GET /help/list?help_search_subject&pattern
Log	log	PUT /log/ "log_level=INFO/WARN/ERROR/FATAL, message:Starting traffic"
perform	perform	POST /perform/ “command:&commandName, optionals...."
	restCommand	GET/POST/PUT/DELETE url body

Category	STC Session Commands	REST API
Sequencer Commands	"Sequencer control commands" on page 1067 "Sequencer action commands" on page 1068	POST /perform/ "command"
Non-sequencer Commands	"Non-Sequencer action commands" on page 1069	POST /perform/ "command"

Converting Spirent TestCenter Test cases to STC REST Test cases

iTest integrates with STC REST API, which you can use to work with the Spirent TestCenter Chassis instead of the STC sessions. iTest provides a conversion wizard to easily convert your existing STC session test cases to STC REST session test cases for your testing. You may convert existing STC session test cases in these two ways.

- Go to i**Test > tools > Convert to STC REST Test Cases...**

OR

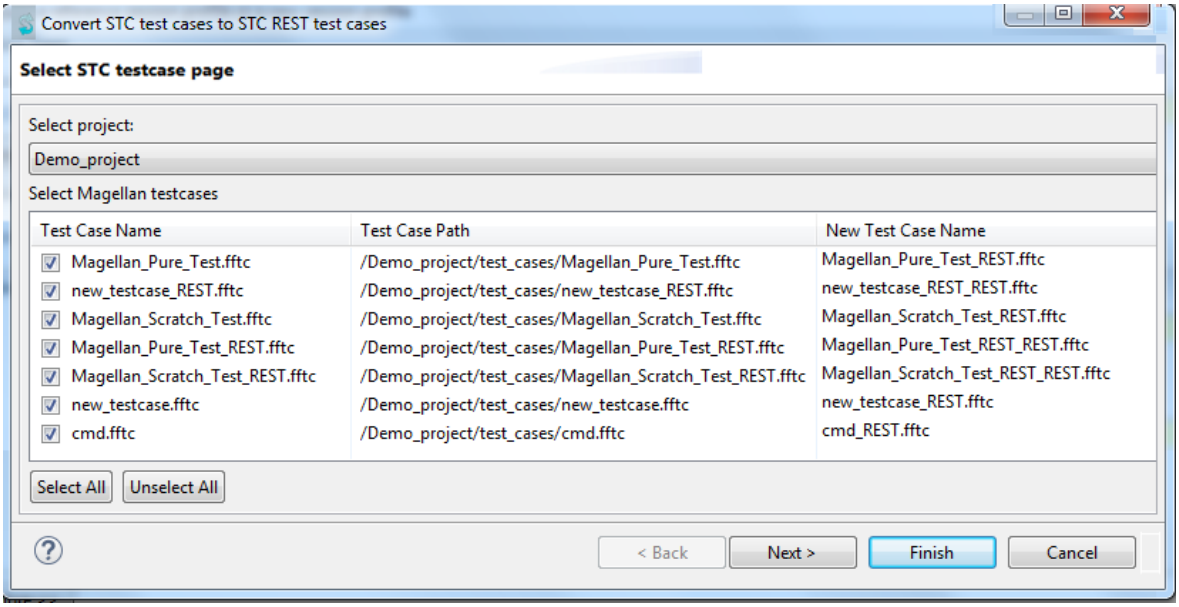
- Go to **iTest Explorer > Test Cases** context menu (right-click on Test cases folder or on any test case within the folder) and select Convert to STC REST Test Case.

Note The iTest Explorer context menu option will be enabled only when you select one or more folders or test cases within a project.

Conversion is a two step process "Selecting STC session test cases" to convert and "Providing STC REST Session Properties" settings as follows.

Selecting STC session test cases

1 The **Select Magellan testcase page** opens with a list of STC session test cases, test case path, and the proposed new name.

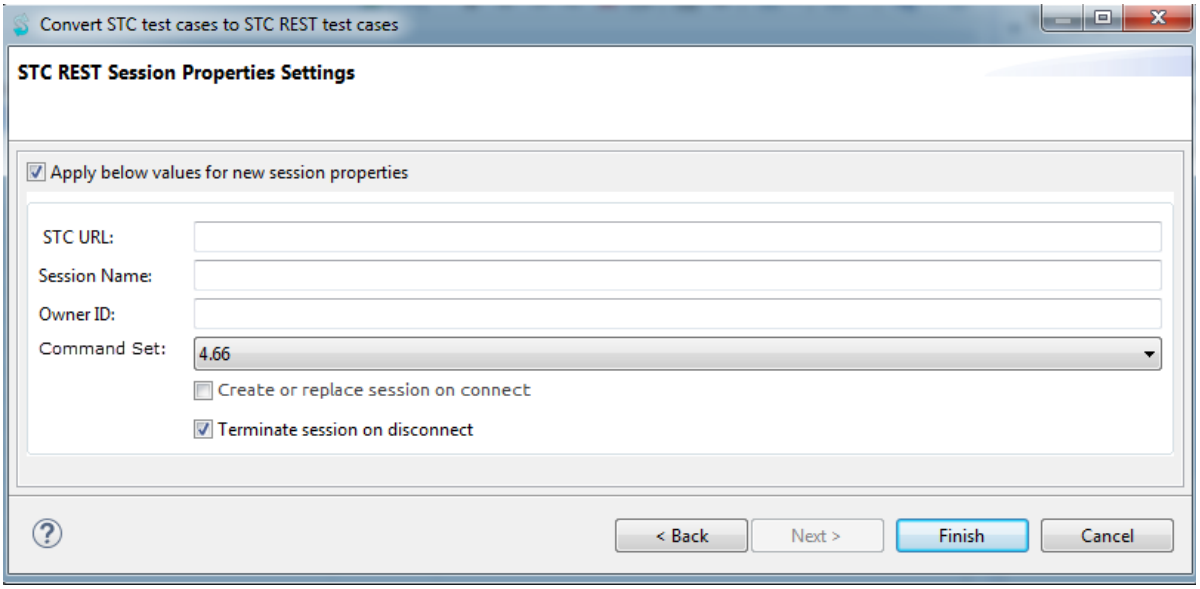


Note The new test case name will include the suffix **REST**.

2 Select test cases to convert and click **Next** to display the **STC REST Session Properties Settings** window

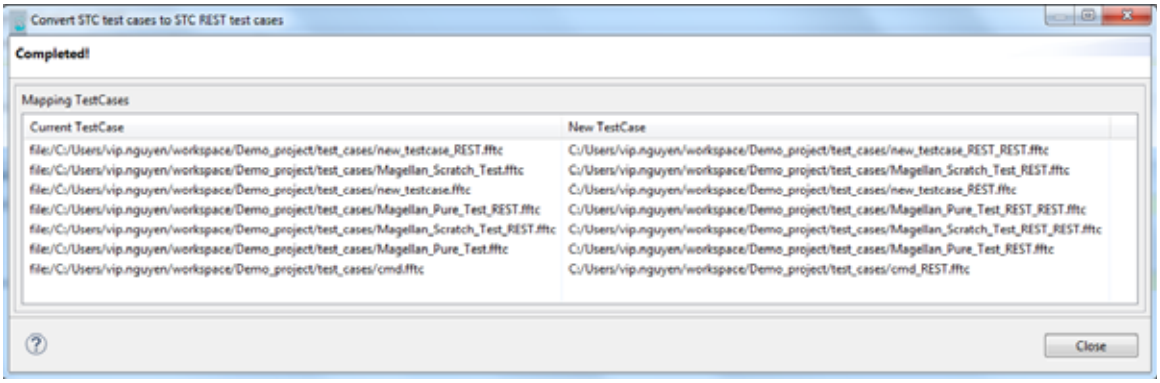
Providing STC REST Session Properties

1 Enter details as described in the table below on the **STC REST Session Properties Settings** window.



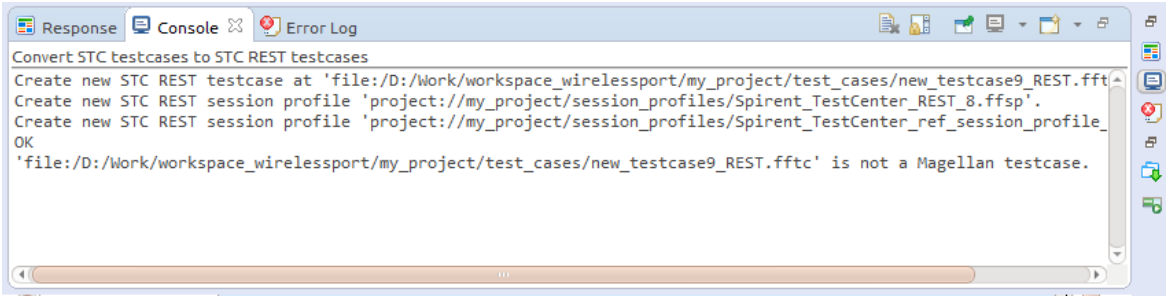
REST Session Properties settings	Description
Apply default value for new session properties	Select to apply default values to the new REST session properties.
STC URL	Enter a valid STC Lad URL. A tool tip shows the valid the URL and port syntax.
Session Name	Enter the session name for the STC REST sessions.
Owner ID	Indicate the owner of the new REST sessions.
Command Set	Select the STC version comamnd set to be used for the REST sessions.
Create or replace new session on connect	Select to indicate that a new session should be created or an existing session should be replaced when connection to the STC Lab is established. Default: unselected.
Terminate session on disconnect	Select to indicate that a session should be terminated when connection to the STC Lab is terminated. Default: selected.

2 Click **Finish** to display list of files converted to STC REST session for your review (**Completed** page).



3 Click Close to exit the window.

Note You may also use the REST commands to convert STC Session test case to STC REST test case via the REST console.



Spirent TestCenter REST session profiles

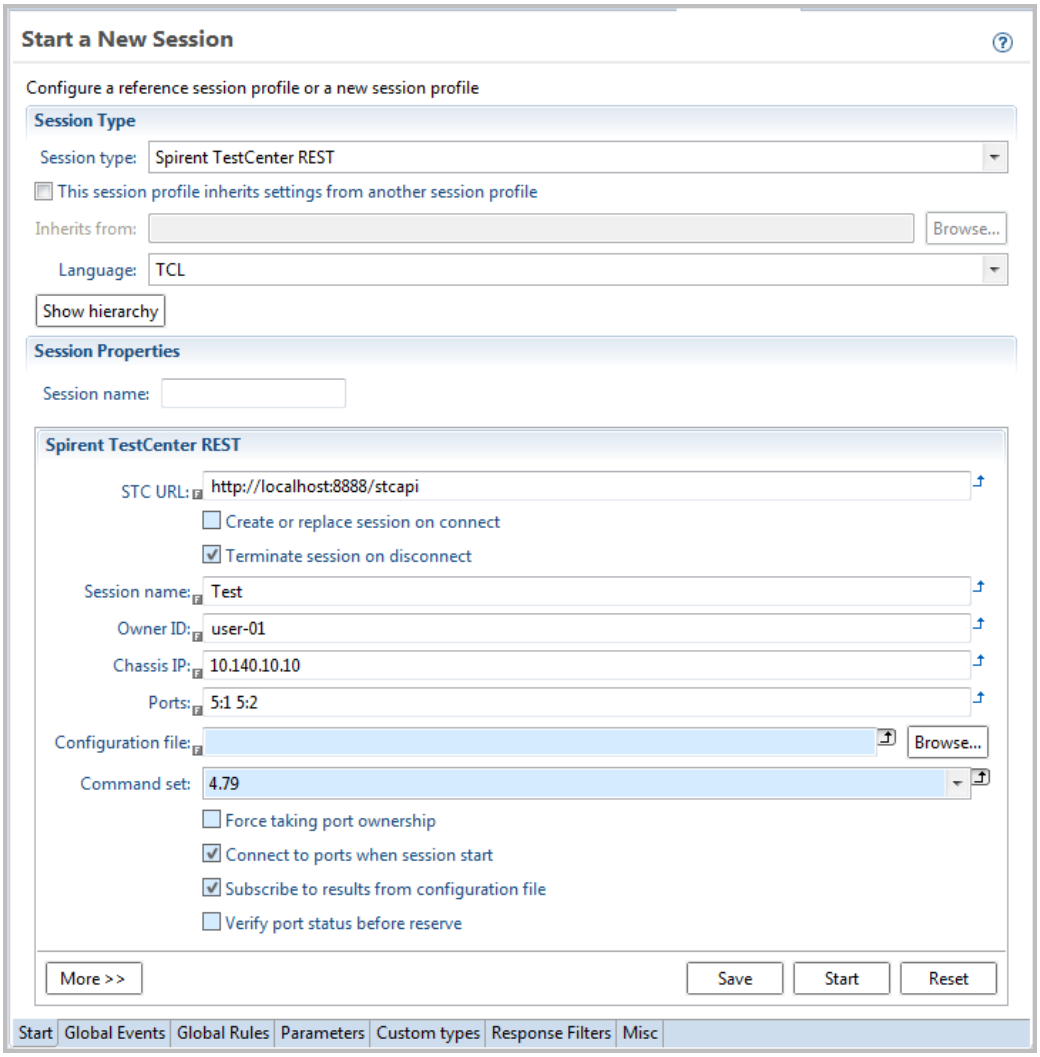
Session profile property settings for Spirent TestCenter REST sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Spirent TestCenter REST session properties



STC URL	(Mandatory): Enter the correct STC Lab server URL (e.g: http://192.168.51.64/stcapi/) or the stcweb app URL. (http://localhost:8888/stcapi).
Create or Replace session on connect	• **When checked**: Creates a new session on STC lab server and replaces an existing session, if any. A message displays if the session already exists. • **when unchecked**: Use the currently running LabServer session for the iTest STC REST session If no session is currently running, an error message displays. **Note** Works in conjunction with the **Terminate session on disconnect** setting.
Terminate session on disconnect	If **checked**, then end the LabServer session when iTest disconnects from LabServer. If **unchecked**, then do not end the LabServer session. Works in conjunction with the **Create new session on connect** setting.

Session name	Name of the session created in STC Lab server.
Owner ID	User ID associated with the session in STC Lab server.
Chassis IP	Specify the IPv4 address or DNS hostname of the device. **Note** If you do not specify a **Chassis IP** value, then you can use **//chassis/slot/port** notation in the **Ports** property to refer to ports on multiple chassis.
Ports	Specify a single port or list of ports for the session. See "To specify a list of port locations" on page 1050. If no **Configuration file** is specified, then, when the session starts, one port is created for each location in the list. The ports are then connected to and reserved. **Note** iTest assigns ports in the listed order. For example, "1:9,1:8" assigns Port 9 first and "1:8,1:9" assigns port 8 first. If you are loading a configuration file and port order is important, you must specify ports in the same order as in the configuration file. **Note** In addition to supporting 10G, 40G, and 100G traffic and port settings, iTest supports you to view and configure 10G, 40G, 100G port settings in TestCenter.console. iTest also supports viewing and displaying of IEEE802.11 port type.
Configuration file	Optional: Specify the configuration file (either XML or tcc format file) to use to configure the device when the session starts. The path is limited to 256 characters. You can generate a configuration file using the TestCenter **configuration save** command. When the session starts: • If no port locations are specified by the **Ports** property, ports in the configuration file with valid locations are connected to and reserved. Offline or inactive ports (**Active** attribute false) are ignored. • If the **Ports** property specifies the same number of port locations as in the configuration file, the ports in the file are mapped to the listed locations in order. • If the **Ports** property specifies fewer port locations than the file, then as many ports as possible are mapped to the listed locations. The mapping begins with the first port in the file and continues until the list is exhausted. All mapped ports are activated and then connected to and reserved. Any ports remaining in the file that have valid addresses and are active are also connected to and reserved. • If the **Ports** property specifies more port locations than the file, then all ports in the file are mapped to listed locations. Mapping begins with the first listed location and continues until all ports in the file are exhausted. All mapped ports are activated. Any locations remaining in the list specified for the **Ports** property are used to create ports. All ports are connected to and reserved.
Command Set	**(Mandatory)** Select the STC command set to be used from the list. The selected vesion of iTest STC session commands will be loaded when the session starts or a testcase runs.

Force taking port ownership	Upon connecting, take ownership so no other user can submit commands.
Connect to port when session starts	When selected, the session connectes to the port automatically when the session starts.
Subscribe to results from configuration file	When selected, you receive a notification of the results of running the configuration file. See "Configuration file" on page 1108 above.
Verify port status before reserve	If selected, iTest will check port status before reserving the ports. If the ports are unavailable, an error displays.

More

Click **More...** complete these relevant information.

- Spirent TestCenter REST: Same information described in "Spirent TestCenter REST session properties" on page 1107.
- To complete relevant information, see also the sections listed in Chapter 53, "Spirent TestCenter sessions".

  "Port Steps, Generator Steps, Custom 16-bit Filter Steps, Stream Block Steps" on page 1094

Misc

Display slot:port format in responses	When iTest returns responses to commands, it can display port references either in **slot:port** format (for example, **2:3**) or in sequential **portIndex** format (just the port identifier, for example, **5**). Check the box to display in **slot:port** format. Uncheck to display in **portIndex** format. Default: unchecked
Use slot:port format in captured steps	When iTest captures responses to commands, it can create test case steps that use port references in either **slot:port** format (for example, **2:3**) or in sequential **portIndex** format (just the port identifier, for example, **5**). Check the box to create command arguments in **slot:port** format. Uncheck to use **portIndex** format. Default: unchecked
Use zero-based stream block indices	Check the box to begin counting stream blocks at 0 (zero). That is, the first stream block is 0. If unchecked, then the first stream block is 1.
Connect time-out (seconds)	(Mandatory) Specify the maximum number of seconds to wait for a session to connect. Default: 180

CHAPTER 78

Spirent Landslide REST sessions

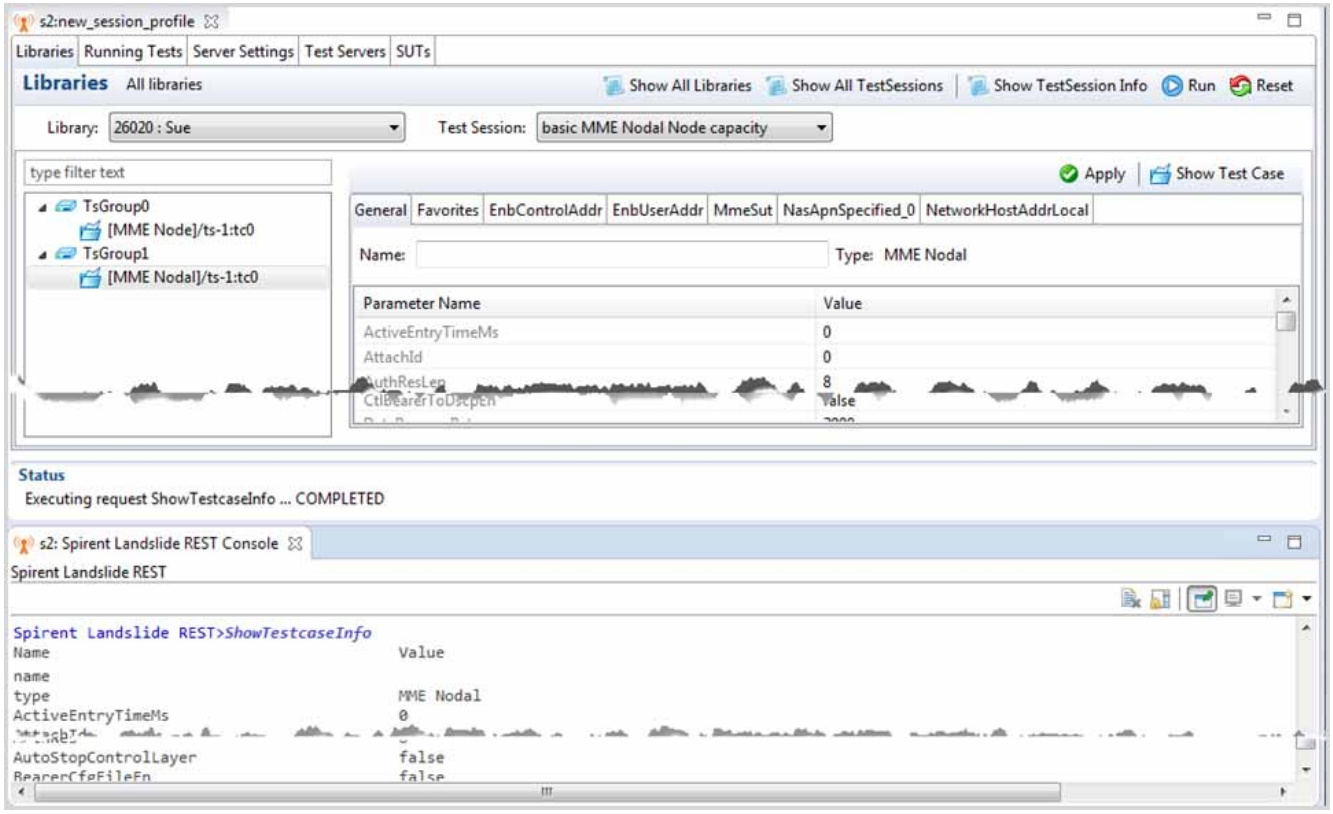
Spirent Landslide REST session window

The session window for Spirent Landslide sessions in iTest has been designed to closely resemble Spirent Landslide. As a result, you can capture Landslide Test Session steps using iTest without having to learn a new interface or new command names. You interact with the iTest Landslide session almost exactly like you interact with Landslide.

iTest integrates with Spirent Landslide and provides REST API to ensure that the automation functionality is readily available to a wide variety of clients using both script and GUI. This integration also eliminates the requirement of a Landslide installation on the client system. This allows you to use existing tools available to create Landslide automation clients that can run on any platform.

- The iTest REST API session communicates with Landslide Test Server via Landslide Lab server using the REST API. So Landslide Lab is required when working with the session.
- The iTest REST session can run on multiple platform without requiring the Landslide libraries.

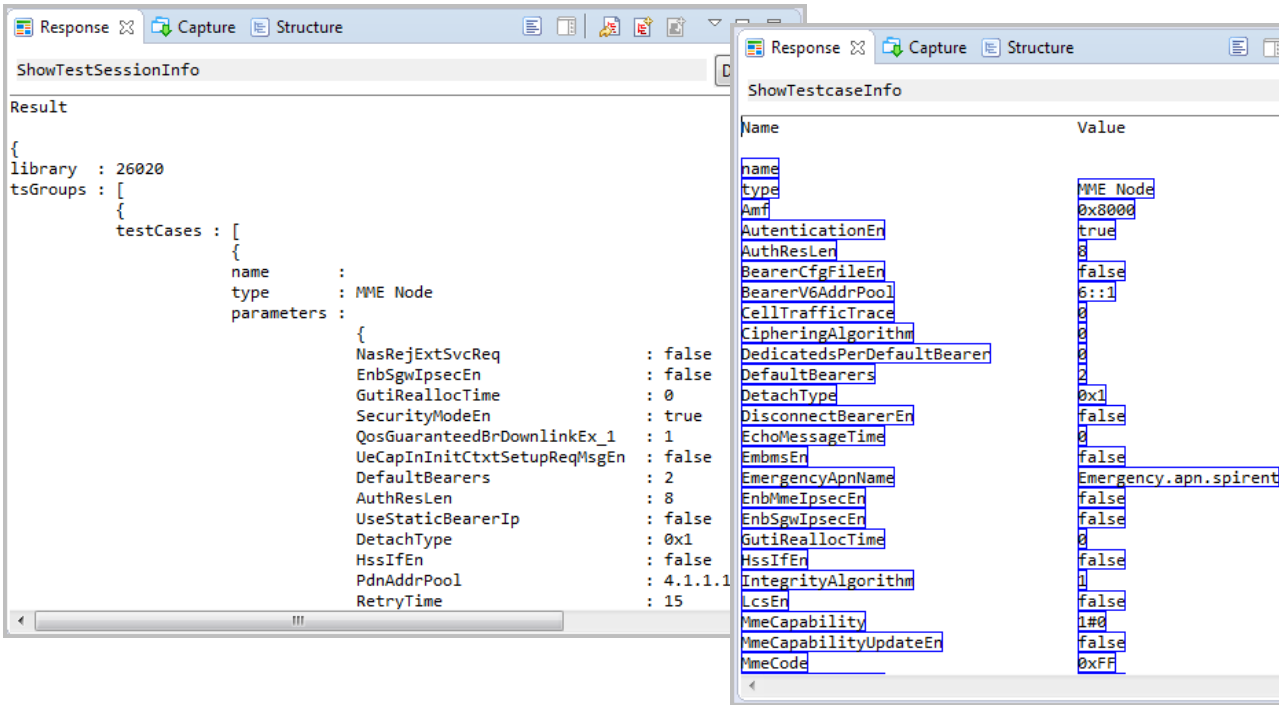
  Since all actions and responses are captured, you can use captured items to create test case steps that configure, control, and request statistics from Spirent Landslide.
- Any action that you perform in the iTest Landslide session is forwarded to Spirent Landslide running on the device. Landslide performs the action and returns its normal response. You can view the response in the **Results** section on the iTest window, just like you do in Landslide.
- iTest captures all of the actions that you perform in a iTest Landslide session and all of the responses returned by Landslide.


s2:new_session_profile
Libraries
Running Tests
Server Settings
Test Servers
SUTs
Libraries
All libraries
Show All Libraries
Show All TestSessions
Show TestSession Info
Run
Reset
Library:
26020 : Sue
Test Session:
basic MME Nodal Node capacity
type filter text
TsGroup0
[MME Node]/ts-1:tc0
TsGroup1
[MME Nodal]/ts-1:tc0
Apply
Show Test Case
General
Favorites
EnbControlAddr
EnbUserAddr
MmeSut
NasApnSpecified_0
NetworkHostAddrLocal
Name:
Type: MME Nodal
Parameter Name
Value
ActiveEntryTimeMs
0
AttachId
0
8
Status
Executing request ShowTestcaseInfo ... COMPLETED
s2: Spirent Landslide REST Console
Spirent Landslide REST
Spirent Landslide REST>ShowTestcaseInfo
Name
Value
name
type
MME Nodal
ActiveEntryTimeMs
0
AutoStopControlLayer
false

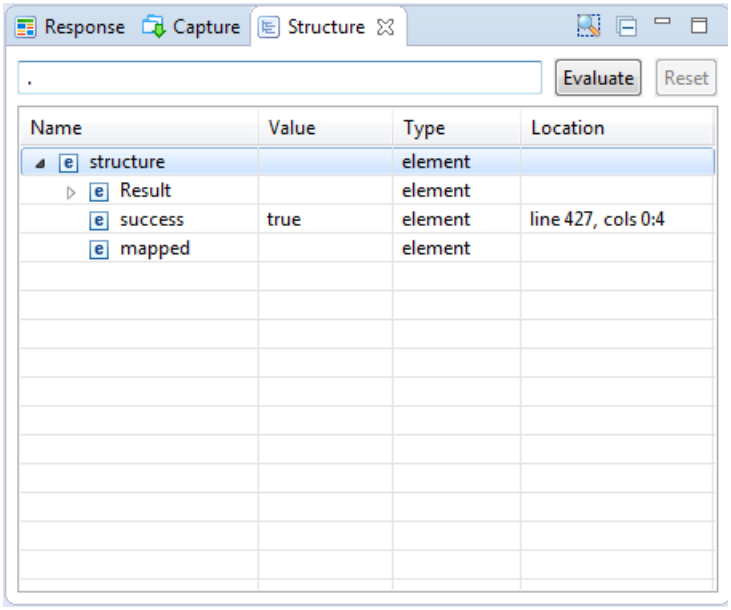
Response mapping

iTest does not use the Landslide protocol to collect or analyze data—other steps in your test case can do that.

iTest saves all responses to iTest Landslide steps as structured data and auto-maps the responses when the captured steps are saved as a test case or added to an existing test case (that is, blue boxes appear in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. Below is the auto-mapped response and structure to a **ShowTestCaseInfo** step in an iTest Landslide REST session.



Note iTest also supports auto-mapped response and structure as a **XML response**.



Spirent Landslide REST sessions

You create a Test case that includes Spirent Landslide REST session in iTest, follow these steps: create and start the Landslide REST session, perform the required actions in Landslide, and save the captured manual steps as a test case.

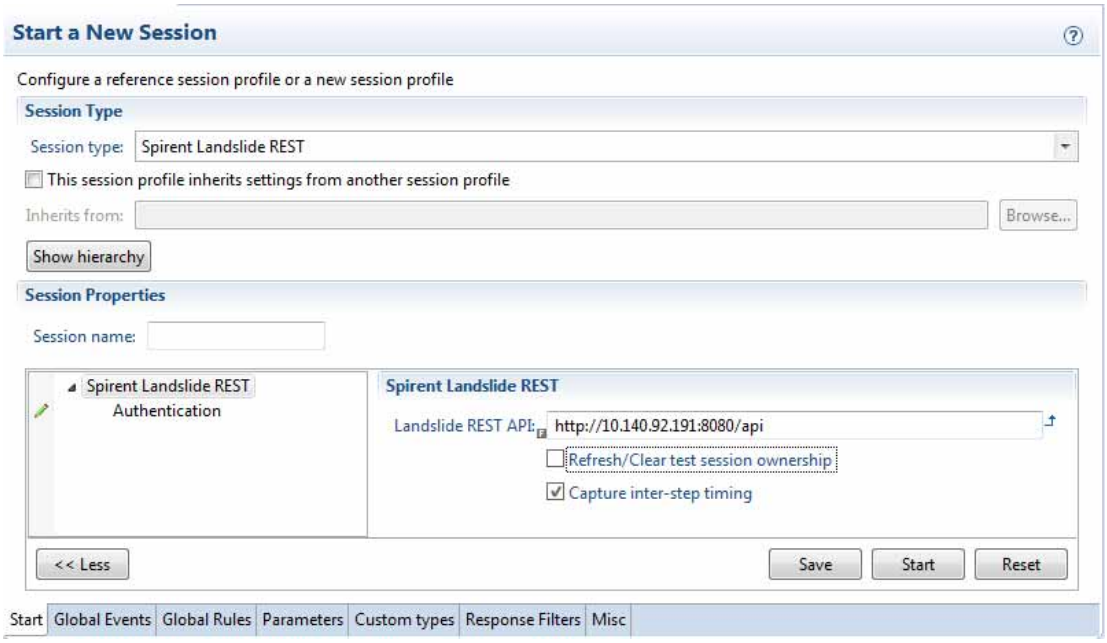
Create and run a Landslide REST Session

1 Ensure that the Landslide REST session profile is properly configured. Specify the Landslide API endpoint and user authentication credentials in the session profile definition as shown below.

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 120. For a detailed description of how iTest uses the settings, see "About property settings" on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.



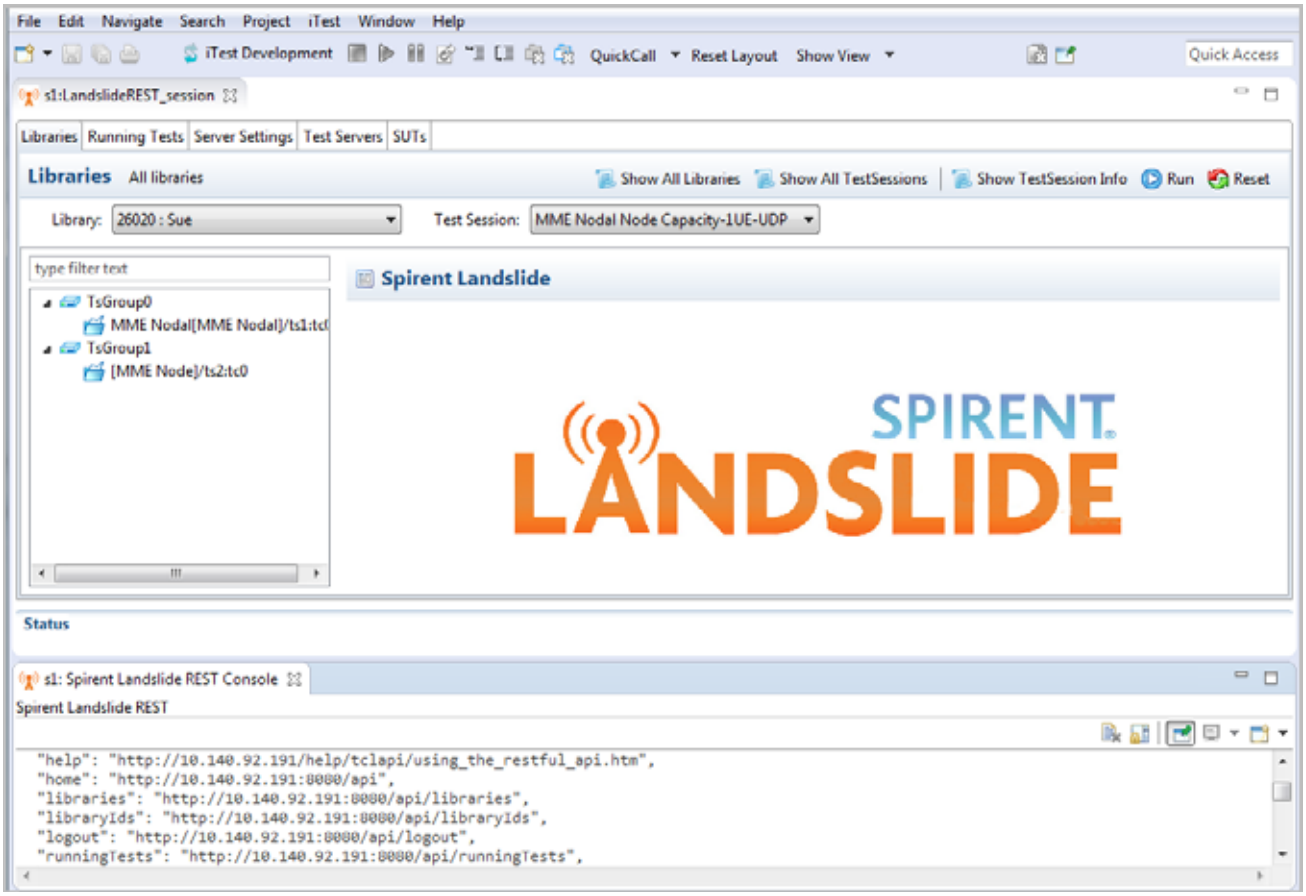
Landslide REST API	Mandatory. Enter the appropriate URL as shown: **https:<tas_Ip_address?:<port>/api**
Refresh/clear Test Session ownership	Default: Not sleeted **Selected:** Test Server will not allow you to run simultaneous tests. An error message will displays and the tests will be aborted. **Not Selected:** The Test Server will abort the test running When running a new test on the Test Server (TS), if the TS is already running test, iTest will abort all running tests on the TS. If all running tests cannot be aborted, the TS will be recycled. The response window will display any error.
Capture inter-step timing	Indicate whether to execute the test case steps with the original time taken to execute each step or execute as fast as possible. **Selected:** Indicates that the rendered test case follows the captured inter-step timing when executing the test case. iTest captures all commands executed in Landslide as steps along with the time taken to execute each step. This time is preserved when the test case is rendered with captured steps and then executed. See "View Landslide Actions steps Added to an iTest Test Case" on page 1464. **Not selected:** Indicates that the rendered test case ***does not*** follow the inter-step timing when executing the test case. That is, run test case steps as fast as it can.
Authentication	Specify authentication: **Basic** Default: **Basic**

User	Required if **Authentication** is set to **Basic** or **Secure**. Specify the user ID for basic HTTP authentication.
Password	Required if **Authentication** is set to **Basic** or **Secure**. Specify the password for basic HTTP authentication.
Trust any SSL certificate from the server	Select to cause iTest (running as the REST client) to trust any SSL certificate coming from the server. Default: Not selected

2 Start the iTest Landslide REST session, The iTest Landslide GUI display as a new window.

Perform the required actions in the iTest Landslide window

iTest will start capturing the actions you perform. You work in the iTest Landslide REST session the same way you work in Landslide (UI editor or Console). When you interact with a Landslide component, iTest performs a Landslide action and captures both the action and the response from Landslide. The Status bar shows the status of all running requests and actions. The REST log console displays all REST request/response messages. See the example below.



1 **Select Library and Test Session**

The Libraries tab allows you to view the list of libraries in the system. You may also view and select the Test Sessions in the selected library. You may view, select, a library, Test Session, override available Test Session's parameters, start a new session with the new parameters.

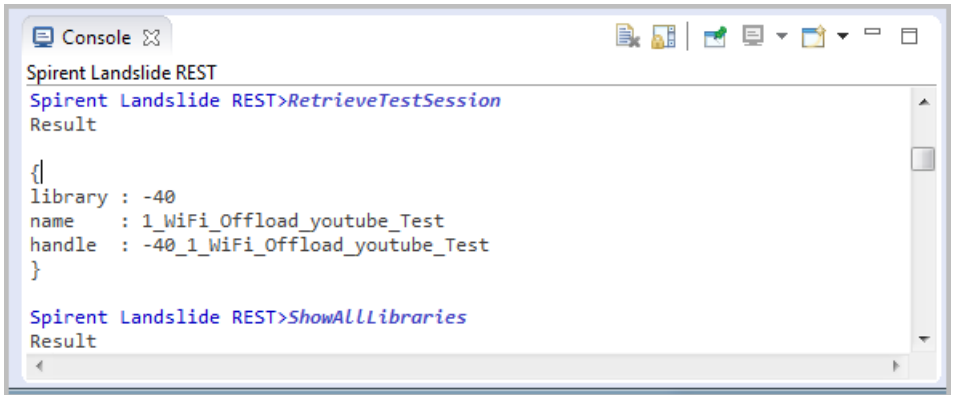
Note Use the buttons on the top left to **Show All Libraries**, **Show All TestSession**, **Show TestSession Info** (the selected TestSession), **Run Test Session** or **Reset Configured Test Session** (discard the changes made). You may click **Apply** to use the changes you made to the Test Session's parameter and Click **Run** to execute the Test Session.

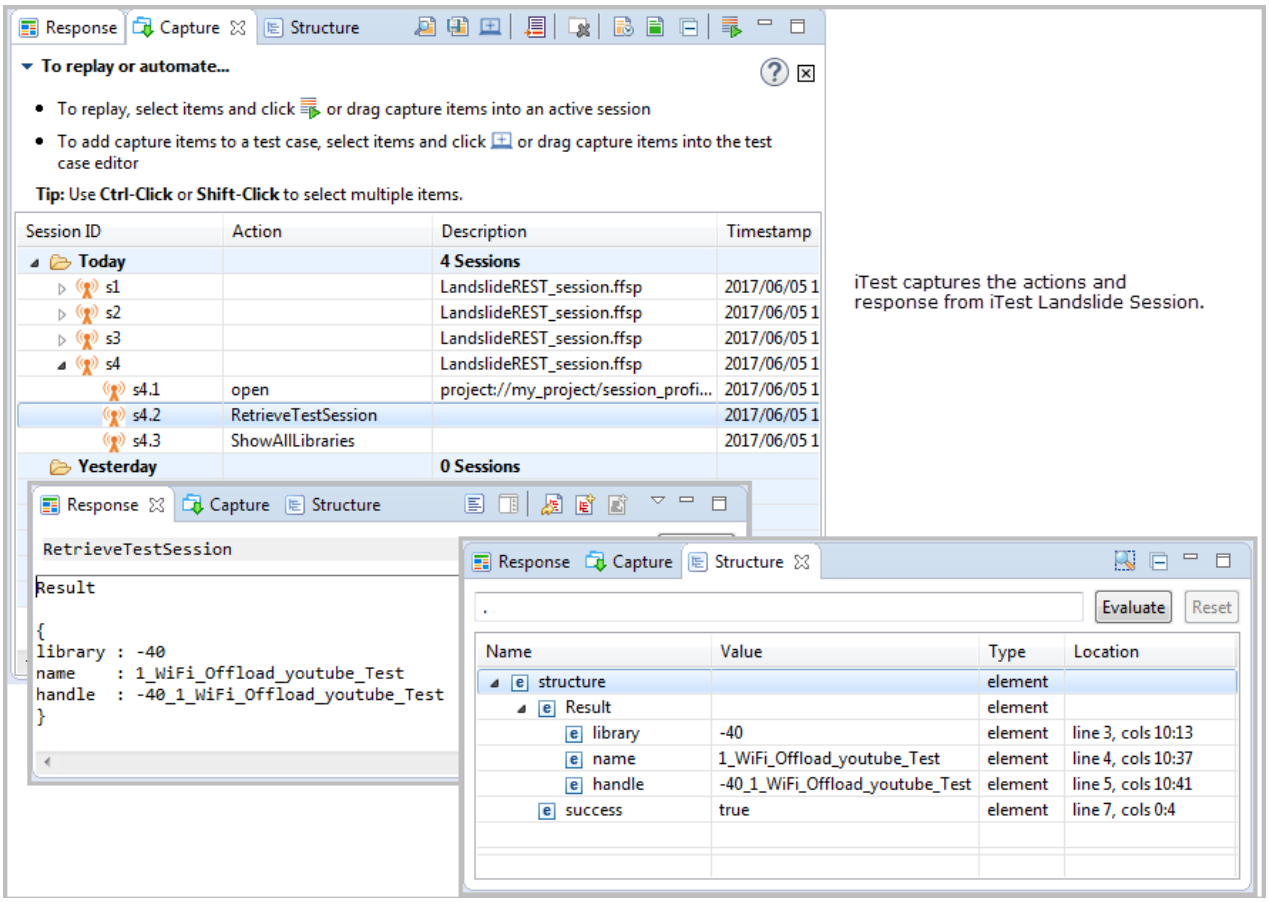
Every action performed in the iTest Landslide UI, is captured by iTest, both the action and the response from Landslide.

Note While not all Landslide commands are available using buttons or other controls on the page, you can perform any Landslide REST command by entering it on the Landslide REST Console (as described in "iTest Landslide Action Commands" on page 1468).

iTest does not support Landslide REST Custom command on Landslide REST Console.

You first select a Library from the drop-down list and select a Test Session in the library. The Landslide REST **Console** displays the command entered and the response from Landslide, which includes the actions you performed. The actions you performed and the response from Landslide are also captured in iTest.





2 View/change Server Settings

The **Server Settings** tab displays a list of all TAS settings. You may changes these settings as required Currently, only the SNMP category is supported.

3 View/change Test Servers

The **Test Servers** tab lists the Test Servers (TS) information and their statuses.

- You may re-assign a TSs for the selected Test Session before running it. However, you cannot change the structure of the Test Session definition. An error displays if you change the structure of the saved Landslide Test Session.

  For example, you cannot change two **TSGroups** to one **TSGroup**. Defining two TSGroups on same Test Session Id is invalid.

- You may change the TS assigned to the test cases. Ensure that you to modify the **TestNodes** correctly, i.e., change the starting IP Addresses, Port selection, etc., to match the new TS.

- You cannot change TS for a Port Reservation test.

4 SUTs (System Under Test Administration Settings)

The **SUTs** tab lists all the SUTs in the system. You may view, create, or modify SUTs as follows:

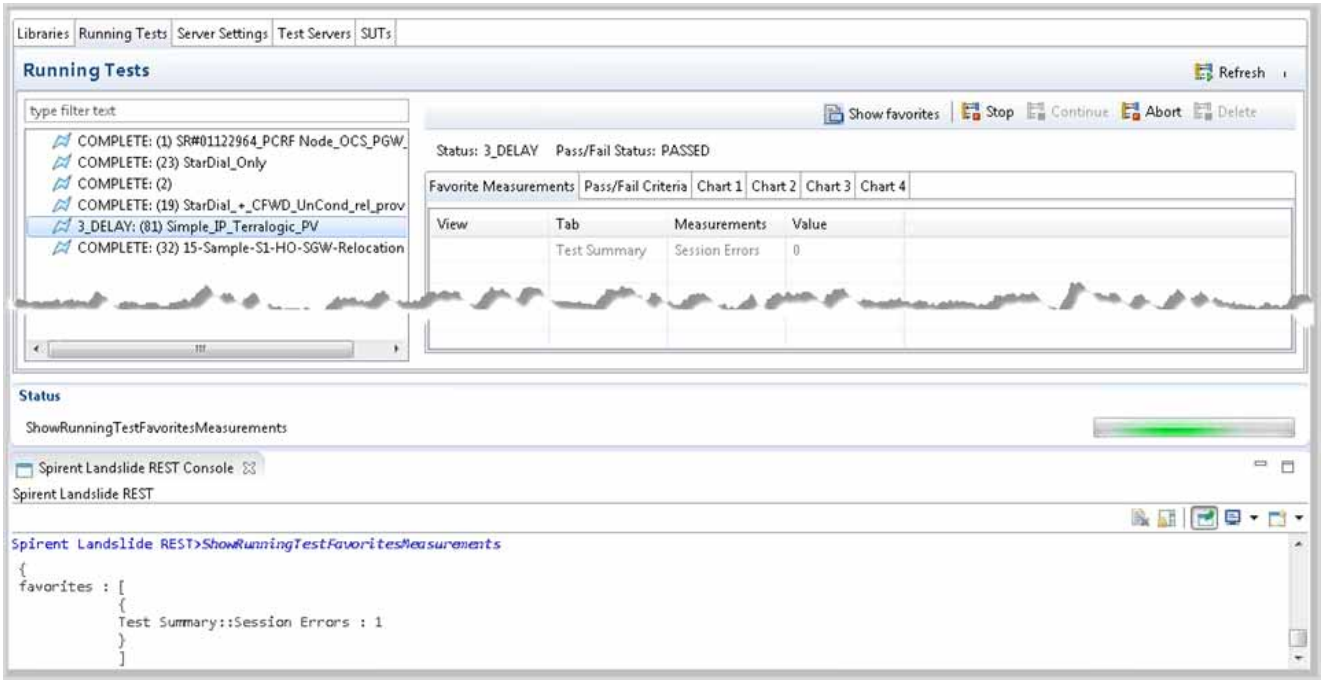
- Modify a selected SUT and click **Apply** to affect your changes.

- Click **Add SUT** and enter the relevant information on the **Add SUT form dialog**. Click **OK** to save the new SUT information or **Cancel** to discard settings.
- You click **Refresh** to view the latest list of SUTs or delete a selected SUT.

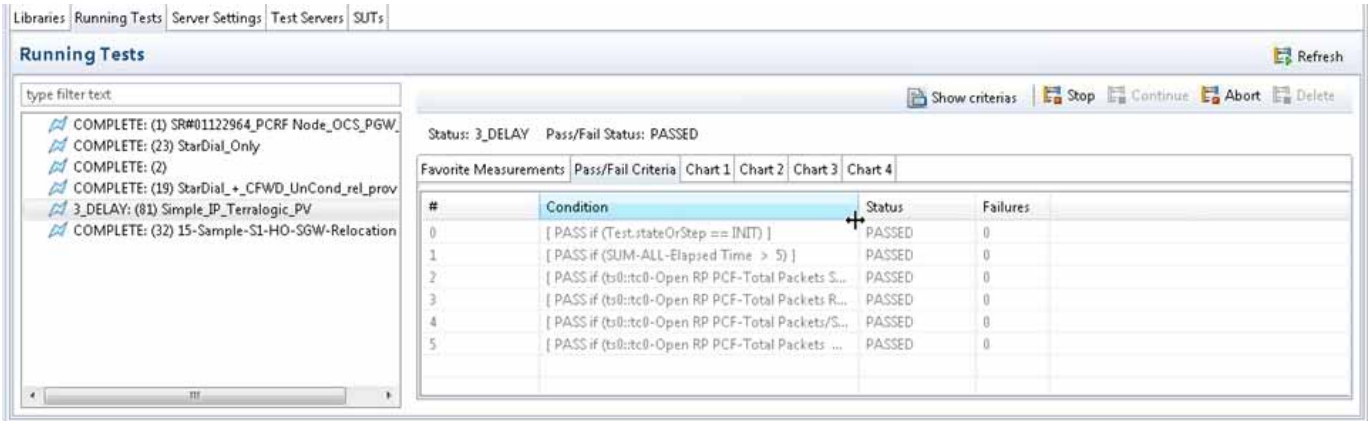
5 Running Tests

The **Running Tests** tab displays the details of live Test Session currently on the TAS (Test Server Administration Server), and allows you to control a Test Session.

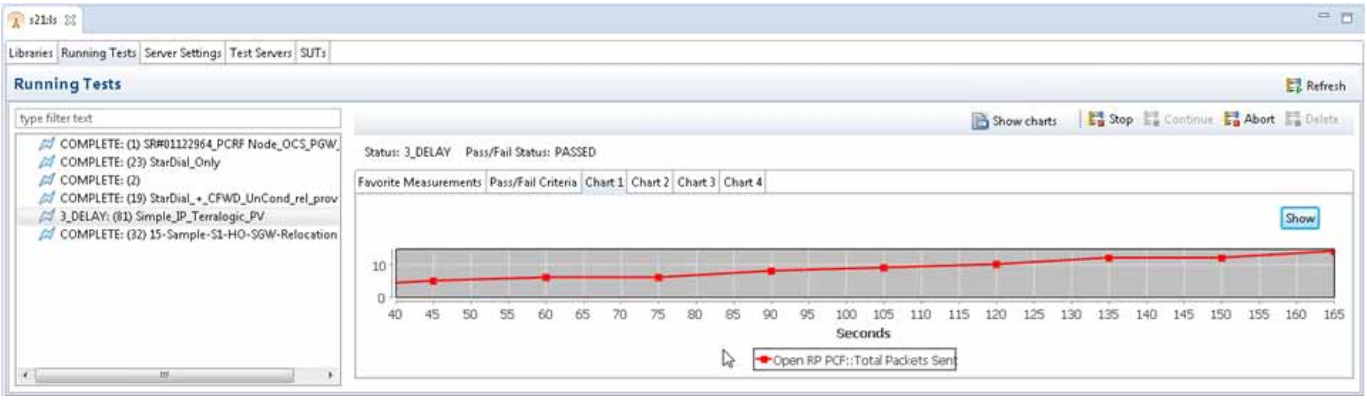
After you have made the required changes for the selected Test Session, click **Apply** and then click **Run** to execute the test. The **Running Tests** tab is populated the run status and results. The results display as per the measurements, pass/fail criteria, and the charts as set up in Landslide.



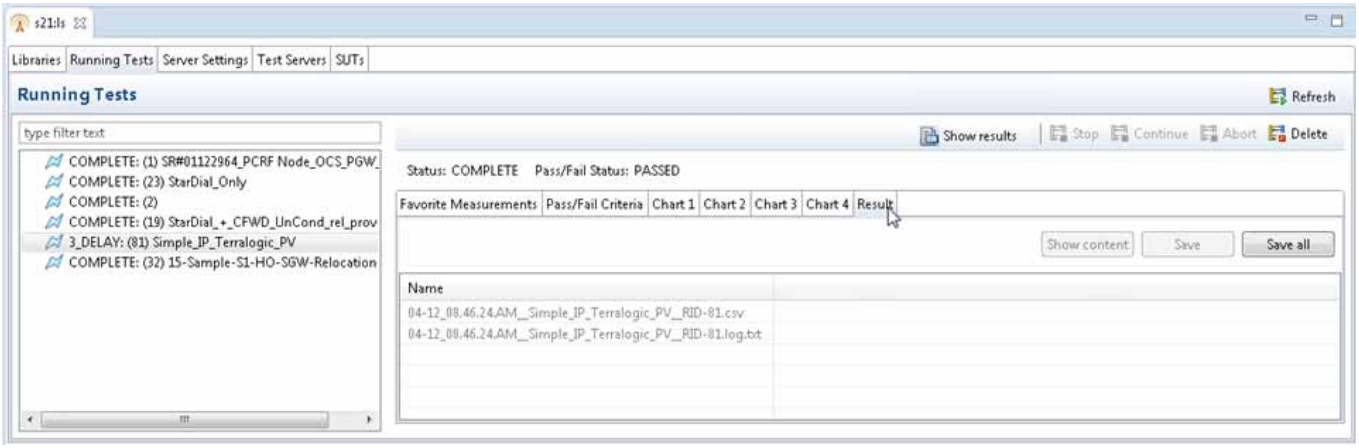
The Pass/Fail criteria displays the settings use to determine the results.



The Charts tab displays as per the measurements set.



After the test completes running, the **Result** tab displays and you may retrieve the end of test results as XLS, CSV, TXT file.

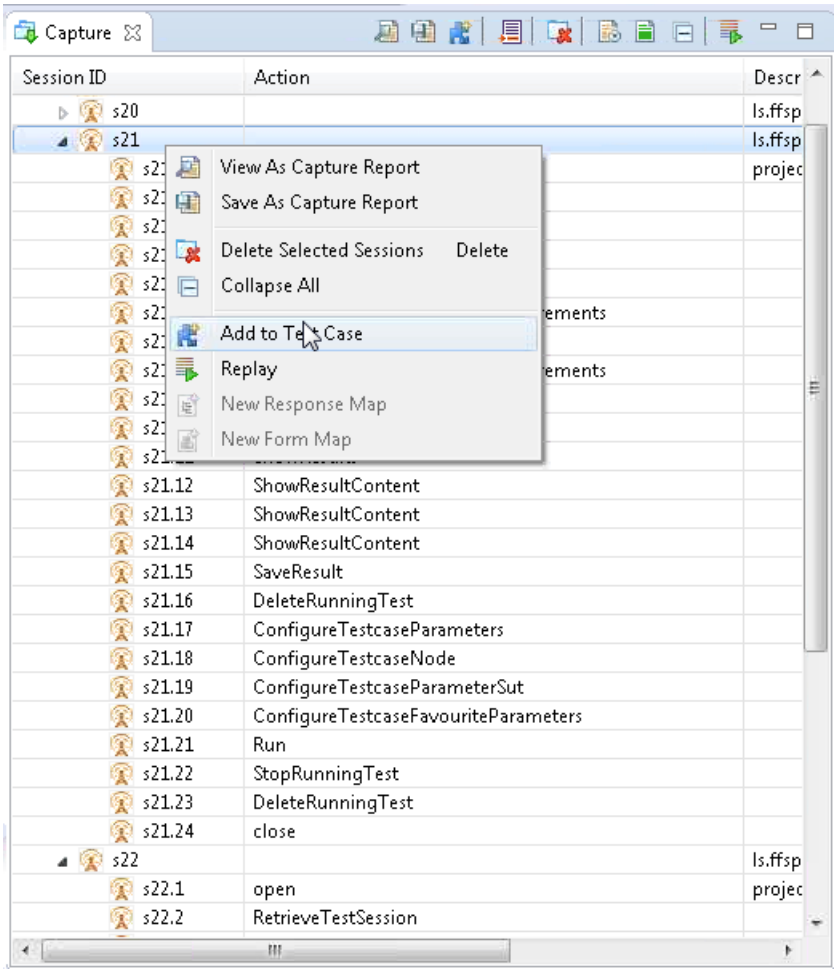


Note iTest does not support viewing or exporting the **.xls** report file generated in Landslide.

- When running a new test on the Test Server (TS), if the TS is already running test, Landslide will abort all running tests on the TS. If all running tests cannot be aborted, the TS will be recycled.
- GET **runningTests**/<ID>/measurements/favorites just returns the value for the last recorded interval or the current interval.
- Result gets specific interval when argument “interval=n”.
- When reconnected to a test (if disconnected for any reason), only the current measurements display, and the results file records the history from the point your reconnect (to the iTest Landslide GUI).
- If you connect at the end of the test, the Charts display with only the last recorded interval.

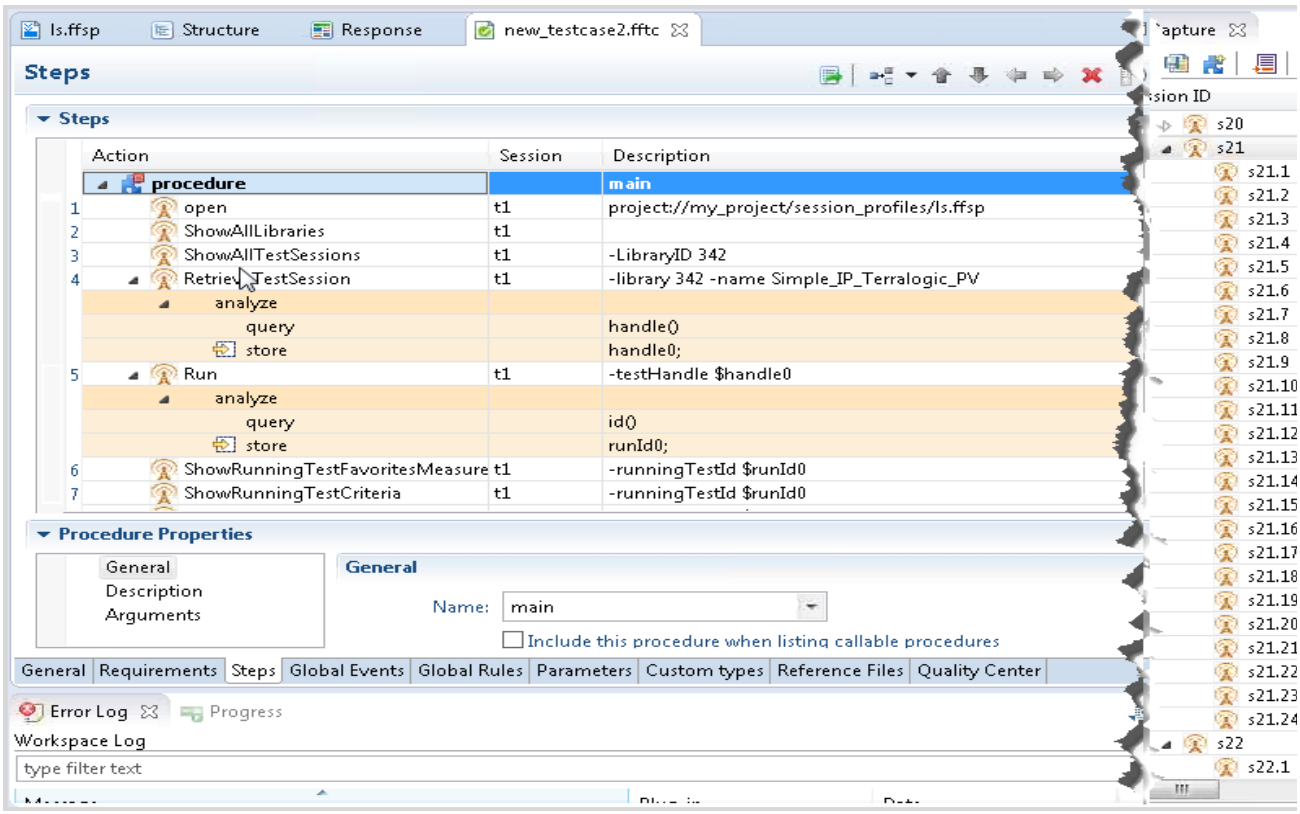
Add Landslide Actions steps captured to a Test Case

When you complete with the iTest Landslide REST Session, select the session and right-click to save the captured steps into a test case.



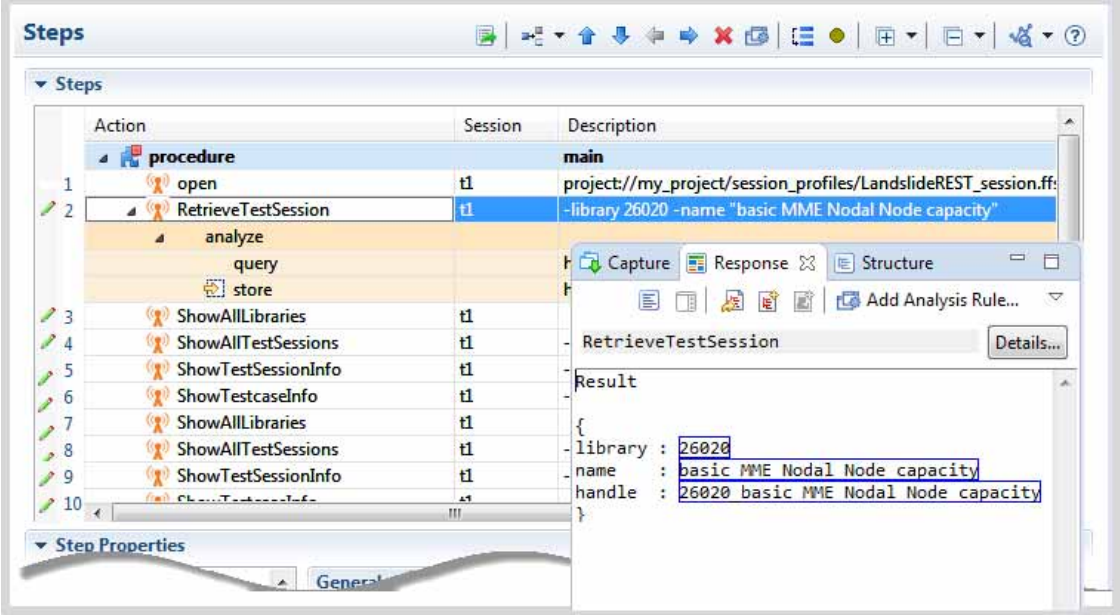
(The **Add Test Case** wizard opens and help you through the process. For details on saving captured steps to test cases, see "Saving interactive steps as steps in a test case: The 'Add to iTest Test Case' wizard" on page 147.)

The Test Case editor opens the new test case.



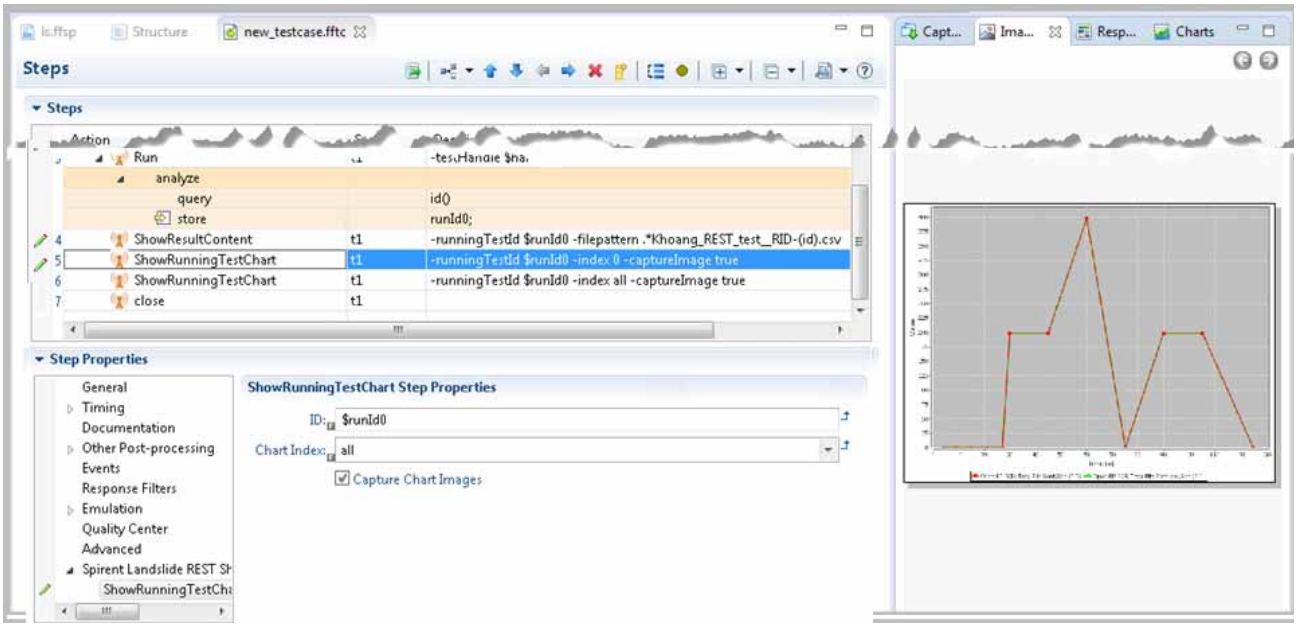
View Landslide Actions steps Added to an iTest Test Case

Select a command step and click the **Response** tab to see the repose map. That is, iTest auto-maps the responses when the captured steps are saved as a test case or added to an existing test case (that is, blue boxes appear in the **Response** view).



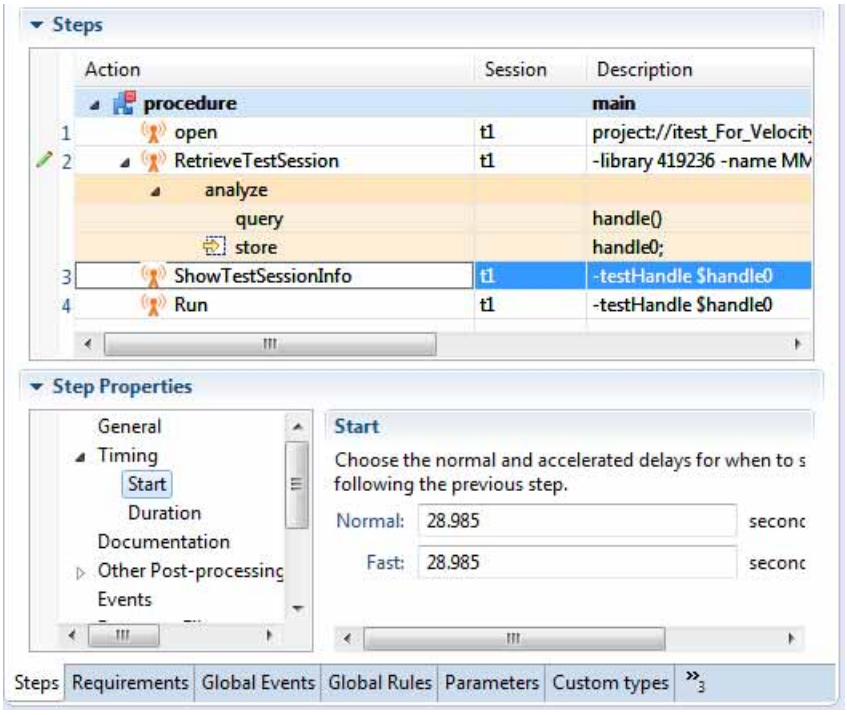
You may view the **Running Tests** charts and result captured in iTest. Go to the **ShowRunningTestChart** action and select **Capture Chart Images**. The chart images gets captured and displays in the iTest Images view, when running the test case. When you export the test report, images will be available in.

The **Capture Chart Images** option is enabled by default while capturing and generating captured step to test case.

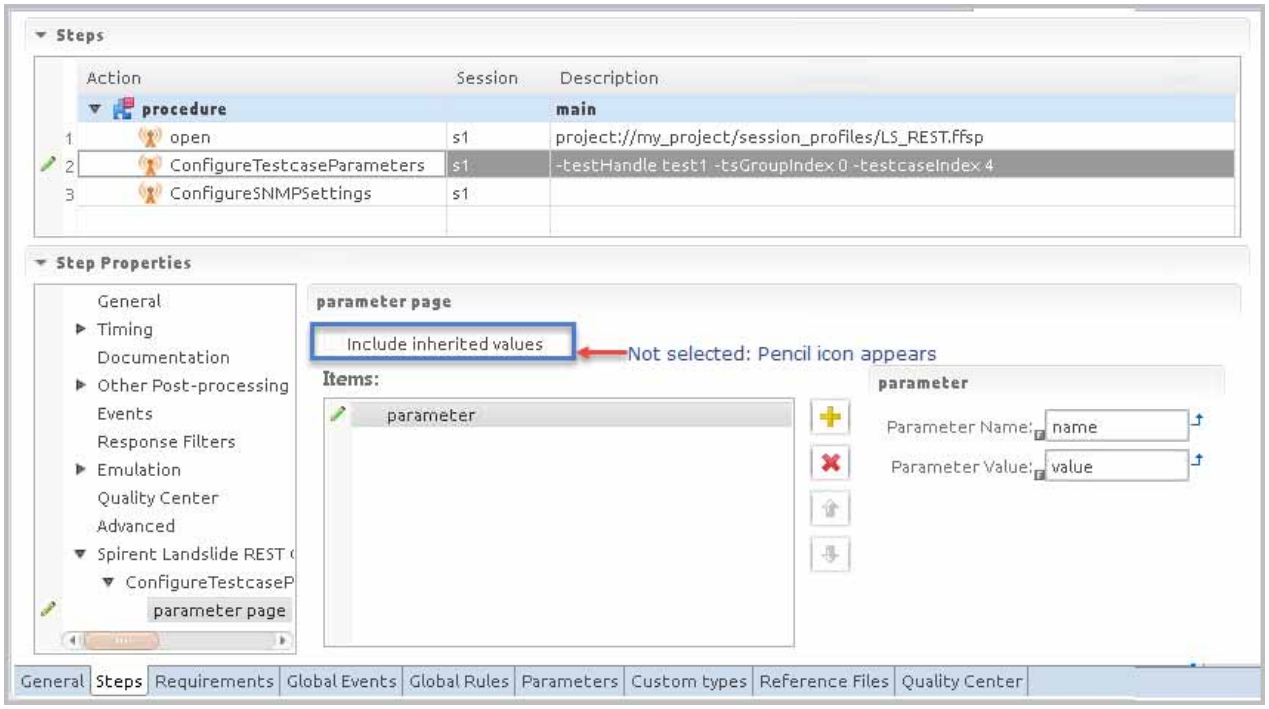


If you enabled **Inter-step timing** when defining the **Spirent Landslide REST** session ("Spirent Landslide REST sessions" on page 1455), the time in-between command execution (in Landslide) is captured as a preference of execution for each step. This time is preserved when the test case is rendered with captured steps and then executed.

Note The default setting will be disabled (no timing delay between steps).



About editing Landslide REST field values and display of the Pencil icon in a Step.



- The Pencil icon does not appear in a Step when any Landslide REST field does not have default value and displays non-inherited value in the step description.

  In addition, the Pencil icon will not display if you input value for Test Handle parameter (which does not have a default value).

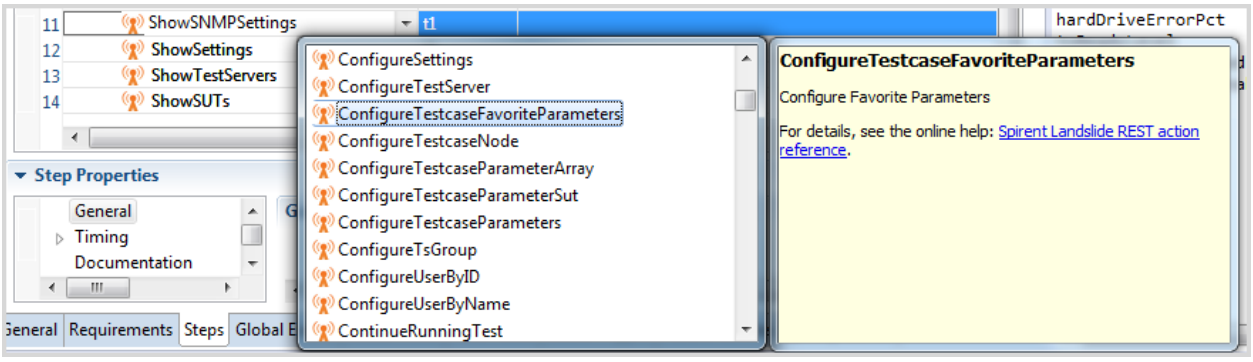
- The Pencil icon does not appear in a Step when a LS REST field includes/displays inherited value in the step description and when the **Include inherited values** is selected.

- The Pencil icon appears in a Step when a LS REST field includes/displays non-inherited value in the step description and when the **Include inherited values** is *not* selected.

1. Continue editing the test case by adding analysis rules and flow-control steps as needed.
   - For steps provide the required property values so that they can be executed properly.
   - For information on working with test cases, see Chapter 7, “Test Cases”.
   - For information on using the Test Case editor, see Chapter 8, “Test Case Editor”.
2. In addition to the actions captured during interactive iTest Landslide session, you may add Landslide REST commands to the test case, if required. To add a step in a iTest Landslide REST test case, click in the **Action** cell (**ctrl + Space**) and select the action from the drop-down list.

Note Any updates made to the Step Properties will update the contents of the Step's description field, and updates to the Step's description field updates the Step Properties. However, this is not support with child property pages.

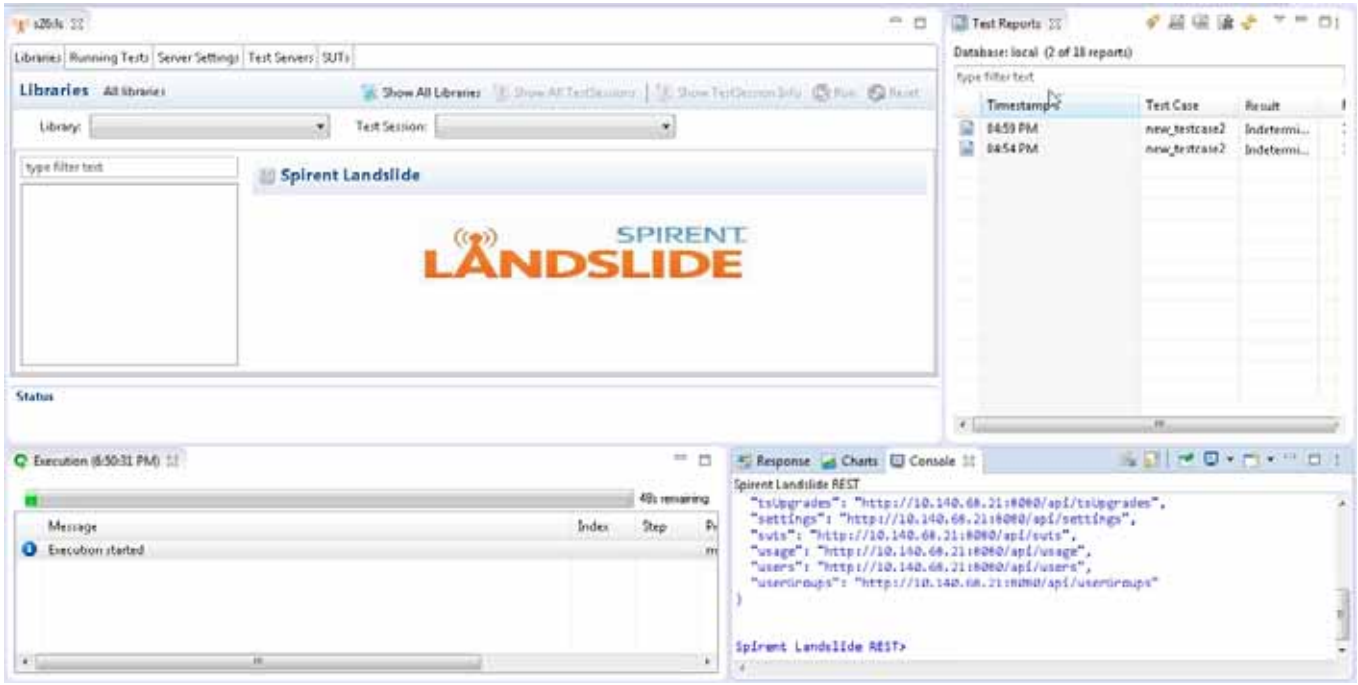
See "iTest Landslide Action Commands" on page 1468 for a list of all commands.



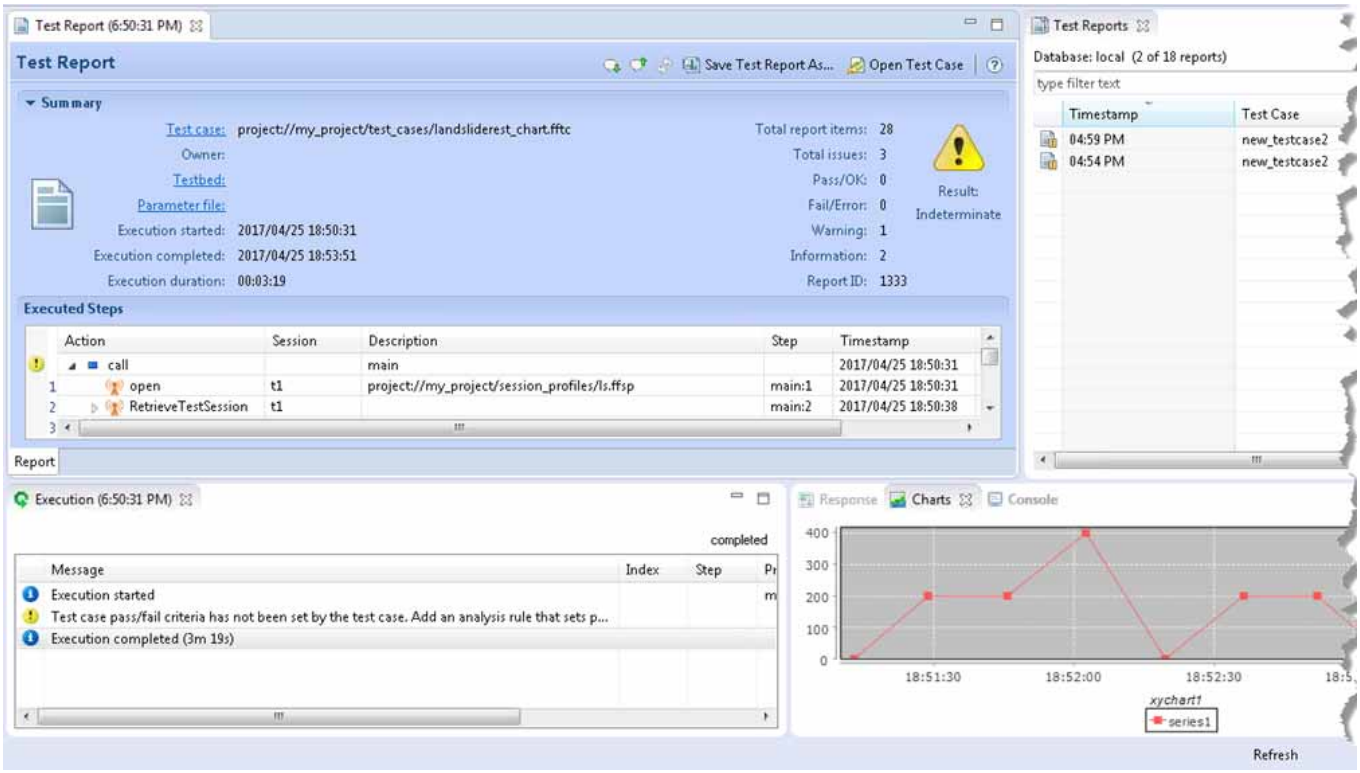
Execute Landslide REST Test Case

You may run the Landslide Actions steps captured to a Test Case as any the other test case. When the execution completed, the test report will be generated.

Open the test case, click Start Execution in New Window. iTest Landslide session opens and starts executing the steps in the test case.



When the execution completed, the test report will be generated.



iTest Landslide Action Commands

This topic describes all Landslide commands in a session with iTest. The actions are captured during interactive Landslide sessions and you can add actions as described in "Add Landslide Actions steps captured to a Test Case" on page 1462.

Group	No	Command	Description
Settings	1	ConfigureSettings	Modifies a particular Category of settings.
	2	ConfigureSNMPSettings	Configure SNMP settings.
	3	ShowSettings	Retrieve a JSON listing of all the settings
	4	ShowSettingsByCategory	Retrieve a JSON listing of settings for the given Category. Currently, 'snmp' is the only category supported.
	5	ShowSNMPSettings	Retrieve a JSON listing of settings for the SNMP Category.

Group	No	Command	Description
Test Servers	6	AddTestServer	Create a new test server
	7	AddTSAssignments	Creates a test server assignment or a list of test server assignments
	8	BackupTestServer	Backs up the test server to the TAS
	9	ConfigureTestServer	Modify Test Server Configuration
	10	DeleteTestServer	Deletes a Test Server by ID
	11	DeleteTSAssignmentsByTestServerName	Deletes all tsAssignments for a given test server
	12	DeleteTSAssignmentsByUserName	Deletes all tsAssignments for a given user
	13	RecycleTestServer	Recycles the test server
	14	RestoreTestServer	Restores the test server from the backup file on the TAS
	15	ShowTestServerInfo	Retrieve a JSON listing of a specific test server by id
	16	ShowTestServers	Retrieve a JSON listing of all test servers
	17	ShowTSAssignments	Retrieve a JSON listing of all tsAssignment user-testServer pairs
	18	ShowTSAssignmentsByTestServerName	Retrieves all the tsAssignments for a given test server name
	19	ShowTSAssignmentsByUserName	Retrieves all the tsAssignments for a given user name
	20	ShowTsUpgrade	Retrieves information for a single test server upgrade file available on the TAS
	21	ShowTSUpgrades	Retrieves a list of the test server upgrade files available on the TAS
	22	UpgradeTestServer	Upgrades the test server using the given upgrade file
SUTs	23	AddSUT	Create a new SUT
	24	ConfigureSUTByID	Modify a specific SUT by SUT Id
	25	ConfigureSUTByName	Modify a specific SUT by SUT name
	26	DeleteSUTByID	Delete a SUT by SUT Id
	27	DeleteSUTByName	Delete a SUT by SUT name
	28	ShowSUTInfoByID	Retrieve information of a specific SUT by SUT Id
	29	ShowSUTInfoByName	Retrieve information of a specific SUT by SUT name
	30	ShowSUTs	Retrieve a list of SUT

Group	No	Command	Description
Usage	31	ShowTestServersUsage	Retrieves a usage and utilization report of the TAS for test server(s)
	32	ShowUsage	Retrieves a usage and utilization report of the TAS
	33	ShowUsageByTestServerName	Retrieves a usage and utilization report of the TAS for a specific test server
	34	ShowUsageByUserName	Retrieves a usage and utilization report of the TAS for a specific user
	35	ShowUsersUsage	Retrieves a usage and utilization report of the TAS for user(s)
Users	36	AddUser	Create a new user
	37	ConfigureUserByID	Modify a user by user id
	38	ConfigureUserByName	Modify a user by user name
	39	DeleteUserByID	Delete a user by user id
	40	DeleteUserByName	Delete a user by user name
	41	ShowUserInfoByID	Retrieve information of a specific user by user id
	42	ShowUserInfoByName	Retrieve information of a specific user by user name
	43	ShowUsers	Retrieve a list of users
Running tests	44	AbortRunningTest	Abort a Test Session
	45	ContinueRunningTest	Continue a Test Session
	46	DeleteAllCompletedTests	Deletes all completed Test Session from the TAS
	47	DeleteRunningTest	Deletes a completed Test Session from the TAS
	48	SendOdcRunningTest	Sends an On Demand Command to the Test Session
	49	SendTcCommandRunningTest	Sends a test case command to the Test Session
	50	ShowRunningTest	Retrieve a high detailed JSON listing of the Test Session with given ID
	51	ShowRunningTestChart	Show Running Test Chart
	52	ShowRunningTestCriteria	Queries criteria of a Test Session
	53	ShowRunningTestFavorites	Queries pass fail criteria, favorite measurements and saved charts of a Test Session
	54	ShowRunningTestFavoritesMeasurements	Queries favorite measurements of a running Test Session
	55	ShowRunningTestMeasurements	Queries measurements of a Test Session
	56	ShowRunningTests	Retrieves a list of the running or completed (live) Test Sessions currently on the TAS
	57	StopRunningTest	Stops the Test Session

Group	No	Command	Description
Test sessions	58	ConfigureTestcaseFavouriteParameters	Configure Favorite Parameters
	59	ConfigureTestcaseNode	Configure Testcase Node
	60	ConfigureTestcaseParameterArray	Configure Testcase Parameter Array
	61	ConfigureTestcaseParameters	Configure Testcase Parameters
	62	ConfigureTestcaseParameterSut	Configure Testcase Parameter Sut
	63	ConfigureTsGroup	Configure TSGroup
	64	ResetConfiguredTestSession	Reset Configured Test Session
	65	RetrieveTestSession	Retrieve Test Session
	66	Run	Start a test with handle
	67	ShowAllTestSessions	Show All Test Sessions
	68	ShowFavoriteParameters	Retrieves a list of the Cross-Reference favorite parameters from a saved Test Session
	69	ShowTestcaseInfo	Show TestCase information
	70	ShowTestSessionInfo	Show TestSession information
	71	ShowTsGroupInfo	Show TSGroup information
Others	72	SaveResult	Save Result files
	73	ShowAllLibraries	Retrieves a list of all the libraries on the TAS and their IDs
	74	ShowResultContent	Show Result content by running id and file Pattern
	75	ShowResults	Show Result file list by running id
	76	StartupCode	Submits a license start-up code to the TAS

Custom commands in iTest Landslide REST Test Case

iTest supports adding a **Custom** step for Landslide REST commands in iTest Landslide REST test cases. The **Custom** step allows you add any new REST commands and functionality included in Landslide. That is, the **Custom** step supports Landslide supported REST API—GET, POST, DELETE methods (GET, POST, PUT, OPTIONS, etc.) and the following functionality:

- Allows you to insert a manual REST command in an active session (i.e., in a **Custom** step added to the Landslide REST session).

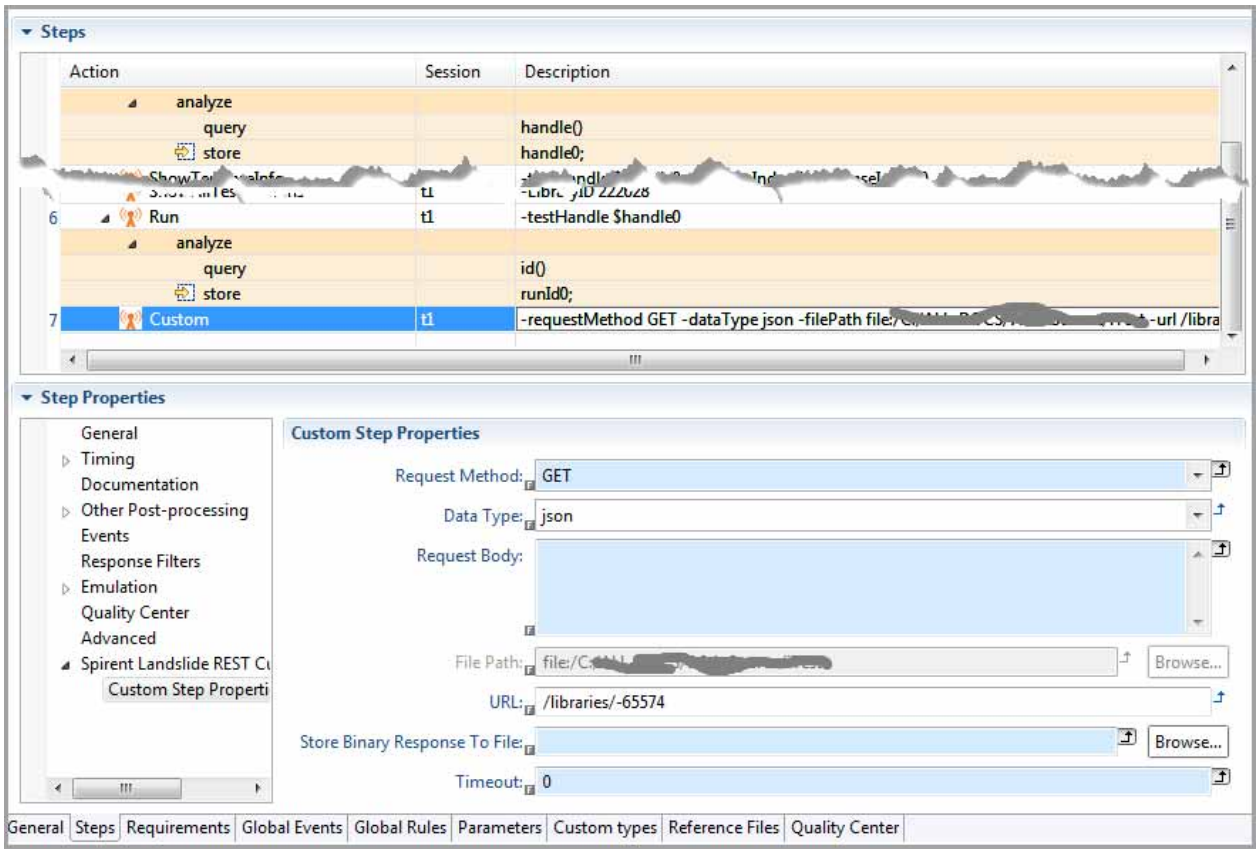
  For example, the Custom step allows you to enter a new Landslide REST command that allows you to type in an API endpoint. The command appends the endpoint to the Landslide REST API URL provided ("Spirent Landslide REST sessions" on page 1455).

  Example: If the URL were: `http://10.108.36.21:8080/api`, and the API endpoint were `favoriteparameters/0/ping.1`, then the command would send `http://10.108.36.21:8080/api/favoriteparameters/0/ping.1` to the Landslide Test Administration Server (TAS).

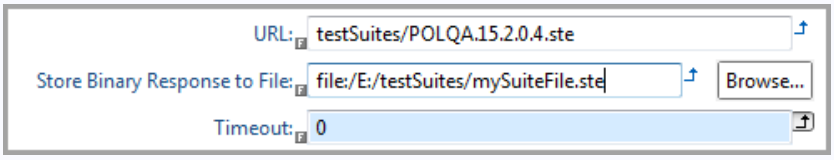
- Provides the necessary variables, such as, auth tokens and test case handles are available for adding the custom REST steps.
- Parses the JSON/XML structure response using the standard iTest XPath parser. Also, allows you use the available XPATH queries to search data in the returned structures.

Custom Command Properties

The following describes the Custom Step properties (**Step Properties > Spirent Landslide REST Custom Properties > Custom Step properties**).



Custom Step Propertiees	Description
Request Method	Values may be GET, POST, or DELETE. Default value is GET. **Note** Landslide REST API does not support methods such as PUT, MOVE, COPY, PATCH, HEAD, etc.
Data Type:	Options: json, XML, binary. Default value is json. **Note** Use **Binary** type to upload file.
Request Body	Enabled only if Data Type is **json or XML**

Custom Step Propertiees	Description
File Path	Enabled only if data type is **Binary**. You may browse to a local file for uploading using the Landslide REST API. FilePath allows you to enter both file location (file://) and project URI (project:// URI) Example: Uploads a Test Data File to the TAS. The file located on your desktop named "myTdf.csv" may be uploaded to the TAS and named as "MyUploadedTdf.csv" on the TAS.
URL	Indicates the API endpoint. For example, if the **Landslide REST API** URL used in the Session profile window ("Create and run a Landslide REST Session" on page 1455) is **http://10.108.36.21:8080/api** and this URL (API endpoint) were **favoriteparameters/0/ping.1** then the command would send the URL as **http://10.108.36.21:8080/api/favoriteparameters/0/ping.1** to the TAS.
Store Binary Response to File	If the response is Binary Data and not text (JSON or XML) response, it indicates that the file a report, pcap, or tdf files are being downloaded from the TAS. Use this argument to provide the file path for storing thedownloaded file (response binary data). Store Binary Response to File supports only with **file:// URI**. Project:// URI is not supported as storing into itar file while running with iTestRT/Agent is not supported. Example: Retrieve a test suite from the TAS. The custom command will download the test suite named **POLQA.15.2.0.4.ste** from TAS and store at path **file:/E:/testSuites/mySuiteFile.ste** 
Timeout	Interger. Allows you to set the timeout (seconds) for the request.

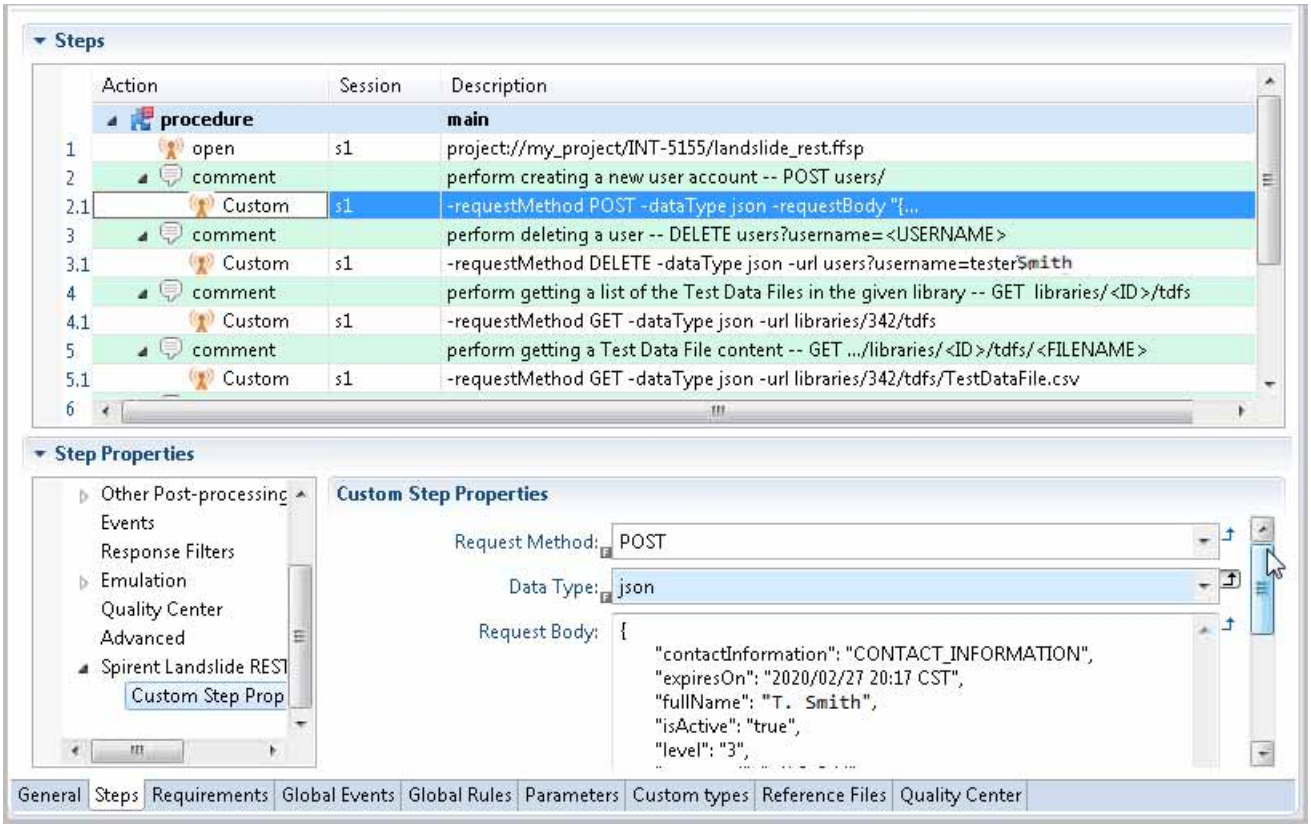
Using Custom Step in iTest Landslide Test Case

This section assumes that you have followed the steps described in the following sections and have added the test case steps.

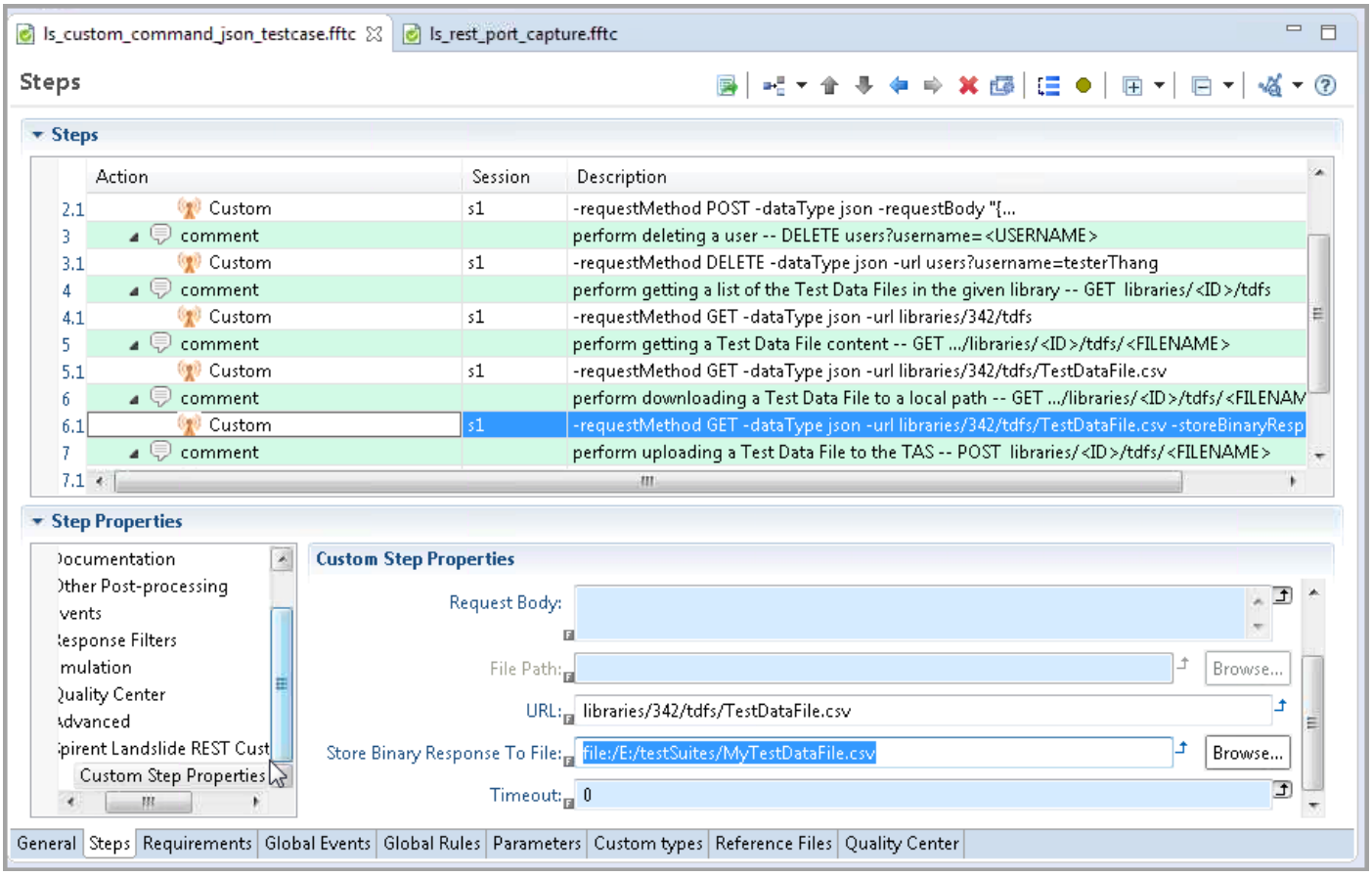
- "Spirent Landslide REST sessions" on page 1455
- "Perform the required actions in the iTest Landslide window" on page 1457
- "Add Landslide Actions steps captured to a Test Case" on page 1462
- "View Landslide Actions steps Added to an iTest Test Case" on page 1464

The example illustrations below shows the **Custom Step** using various TEST API.

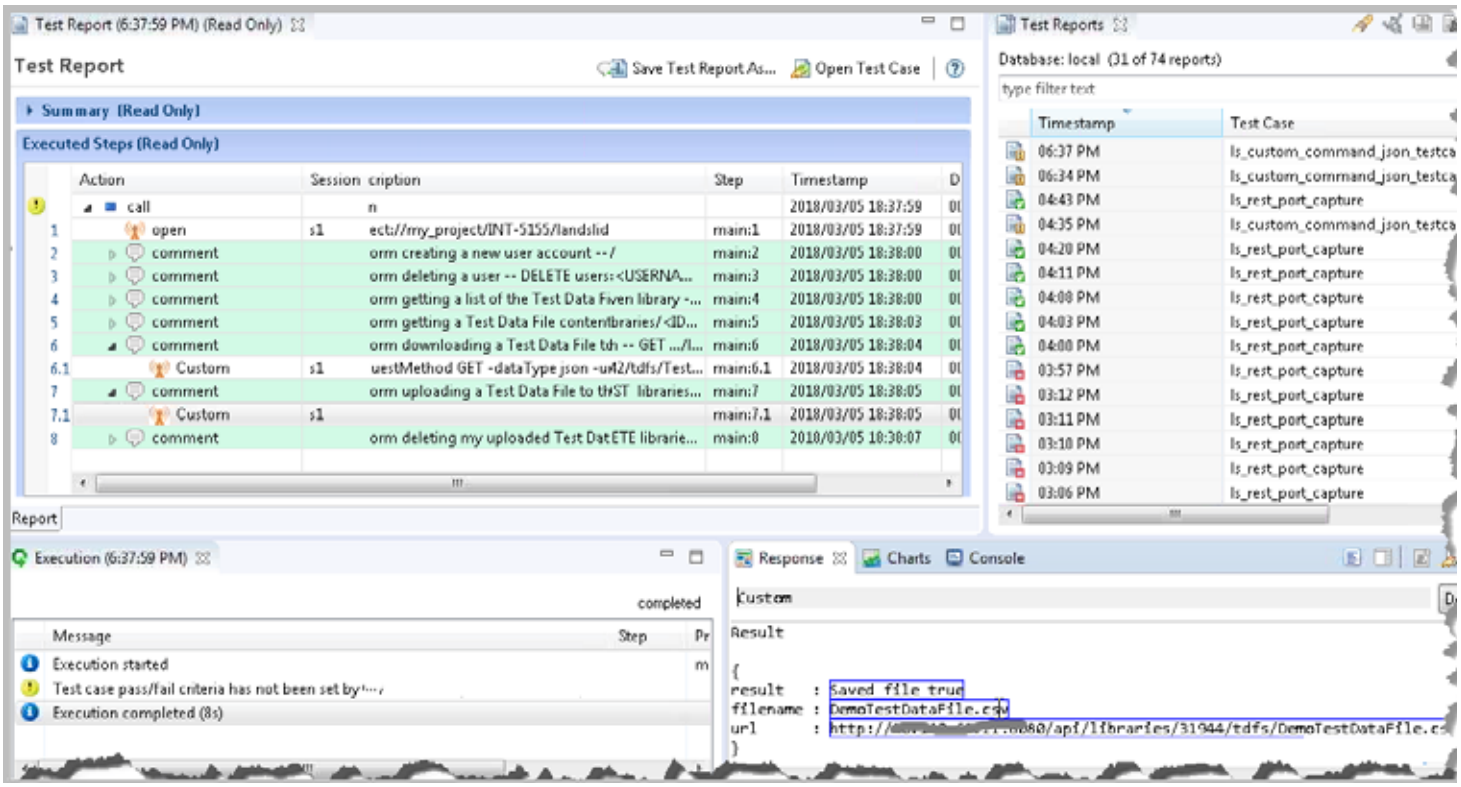
Method POST (to create a user).



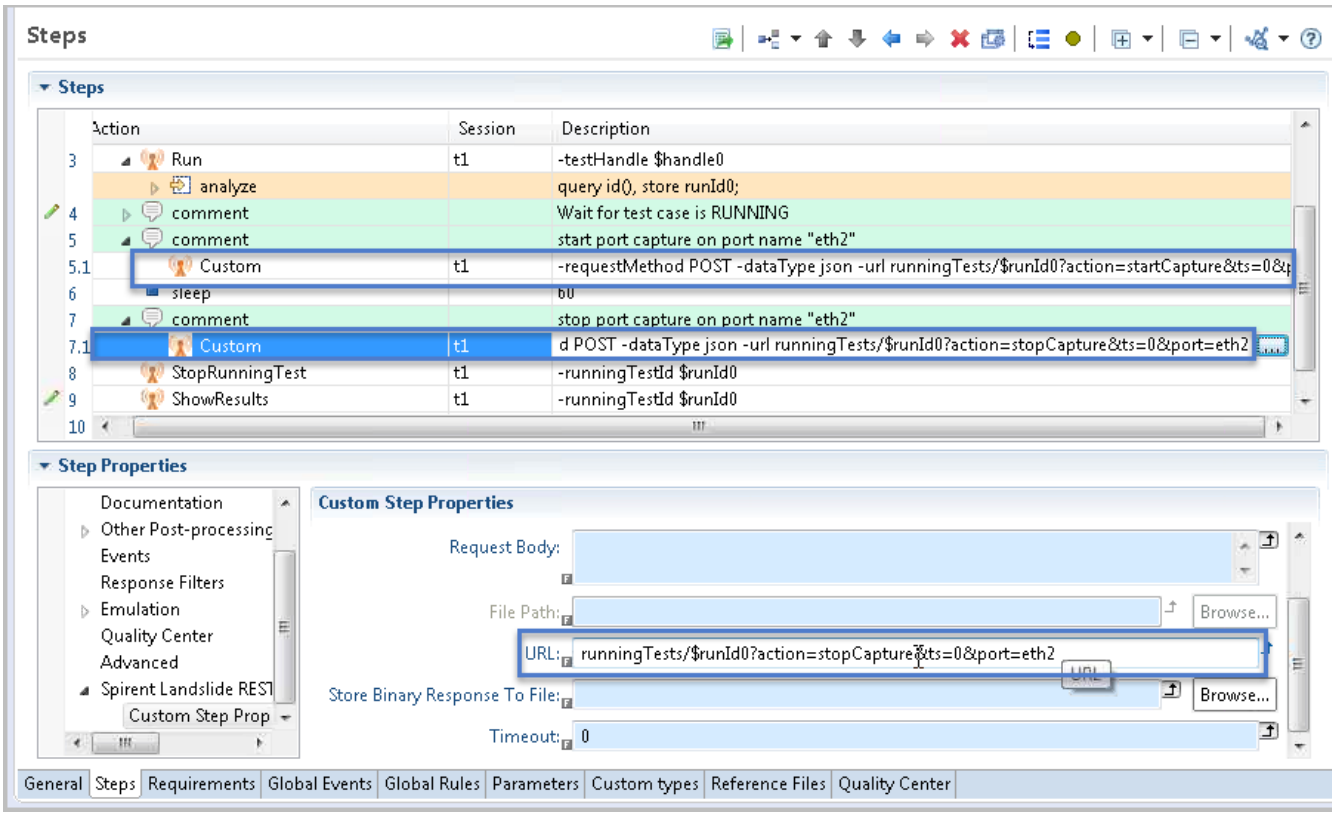
Get—Store binary response to file



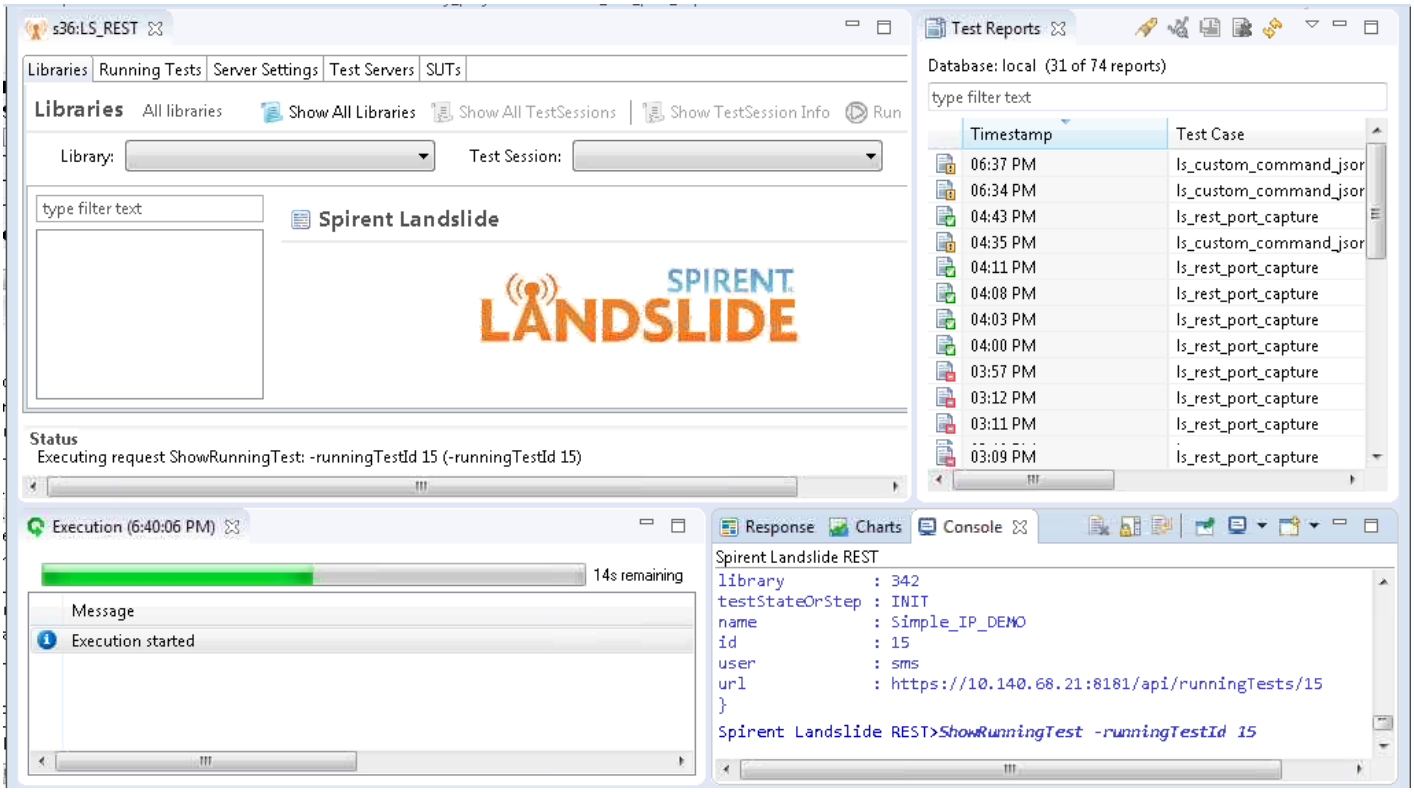
Execution report—Store binary response to file.



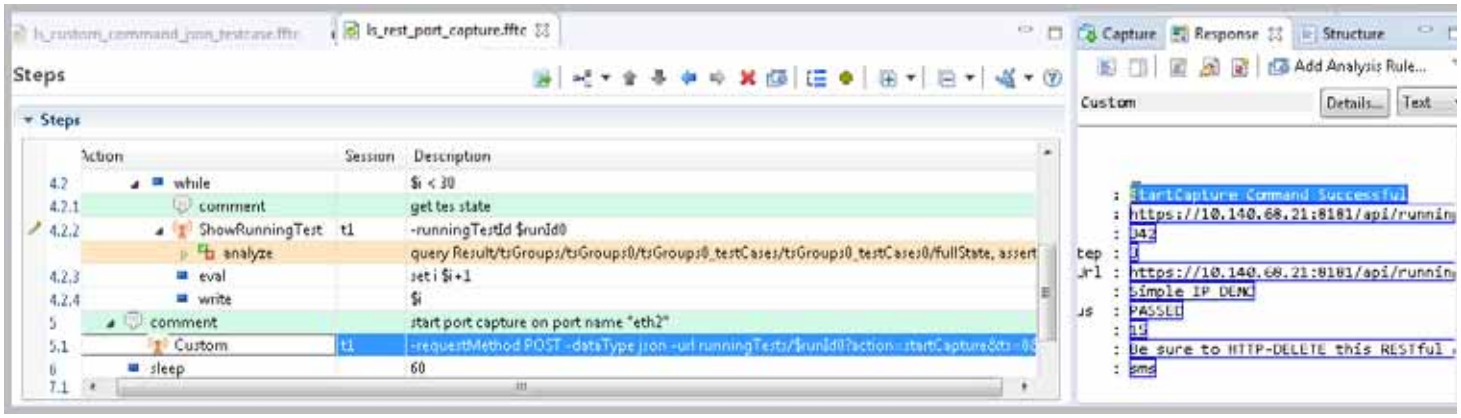
Capture step to Start and Stop port capture

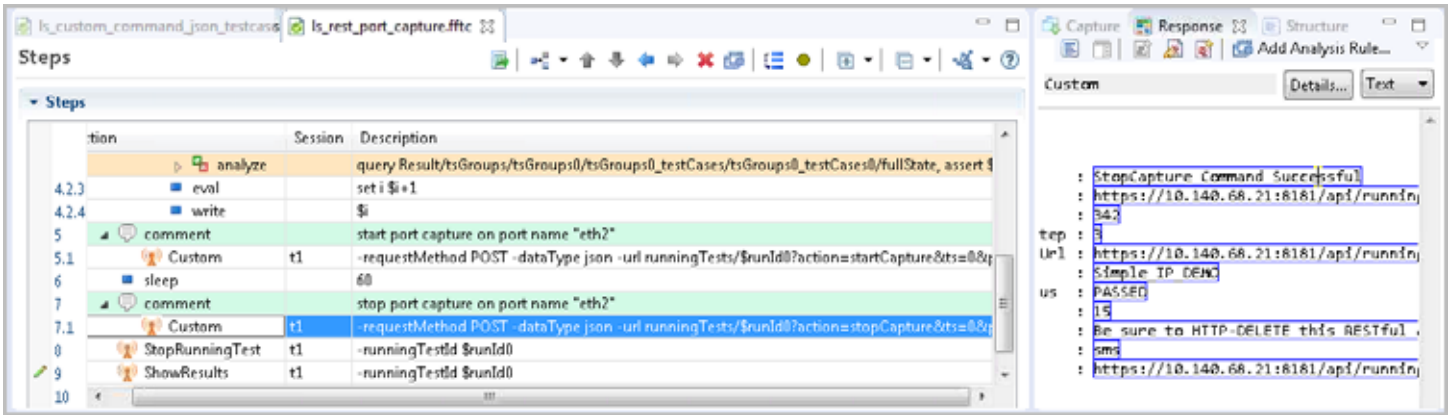


The Start port capture begins when the test runs and the Landslide test execution starts.

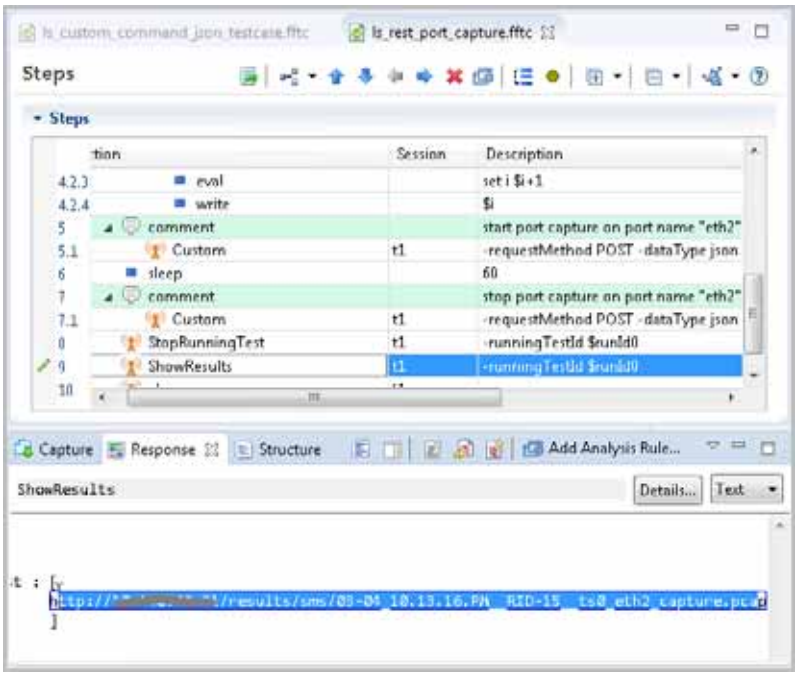


You may verify the successful port capture via iTest test case response view.





The response view of the **ShowResults** command shows that the port capture resulting in a PCAP file.



CHAPTER 59

Python Sessions

Overview

iTest Python session is a terminal session, similar to Tcl Shell. The session uses native Python interpreter to getting responses. Supported versions of interpreters are: 2.4-2.7 and 3.0-3.6.

iTest support an internal Python interpreter and also allows you to point to an external interpreter via Preferences settings (see “Setting preferences for Python” on page 1166).

The Python Terminal sessions allow you to execute commands in the Python interactive shell, which are captured as Steps in iTest. The response of each step will be auto-mapped by a Python-optimized mapper which handles Python language primitives such as:

- **Named constants:** True, False, None
- **Numbers:** Integer - 1, Float - 2.3, Long - 4L, Complex - 5j
- **Strings:** simple - “str”, Unicode - u”str”
- **Tuple:** (1, 2, 3, 'Spam')
- **List:** [0, 1, 2, 3]
- **Dictionary:** {'APT Products': 1, 'iTest': 0}

Create a new Python session profile

Create a Test case that includes Spirent Python session in iTest, following these steps: create and start the Python terminal session, perform the required actions in the terminal session, and save the captured manual steps as a test case.

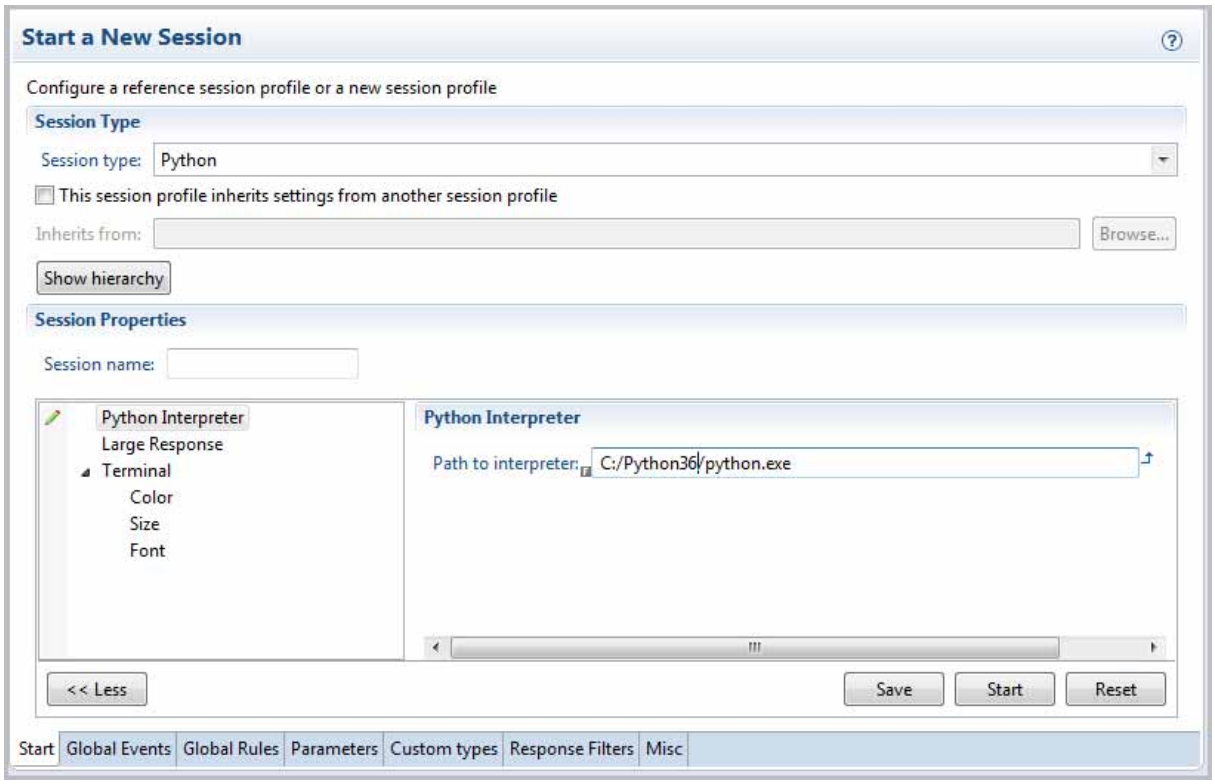
Create and run a Python Session

1. Ensure that the Python session profile is properly configured. Specify the path to Python interpreter, and if required, change the size of the large repose, terminal color, size, and font.

   This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.



Path to interpreter	Enter the path to the interpreter to be used for this particular session. If you do not specify a path to the interpreter, iTest uses the path specified in the preferences setting. See "Setting preferences for Python" on page 1166.

Large Responses

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

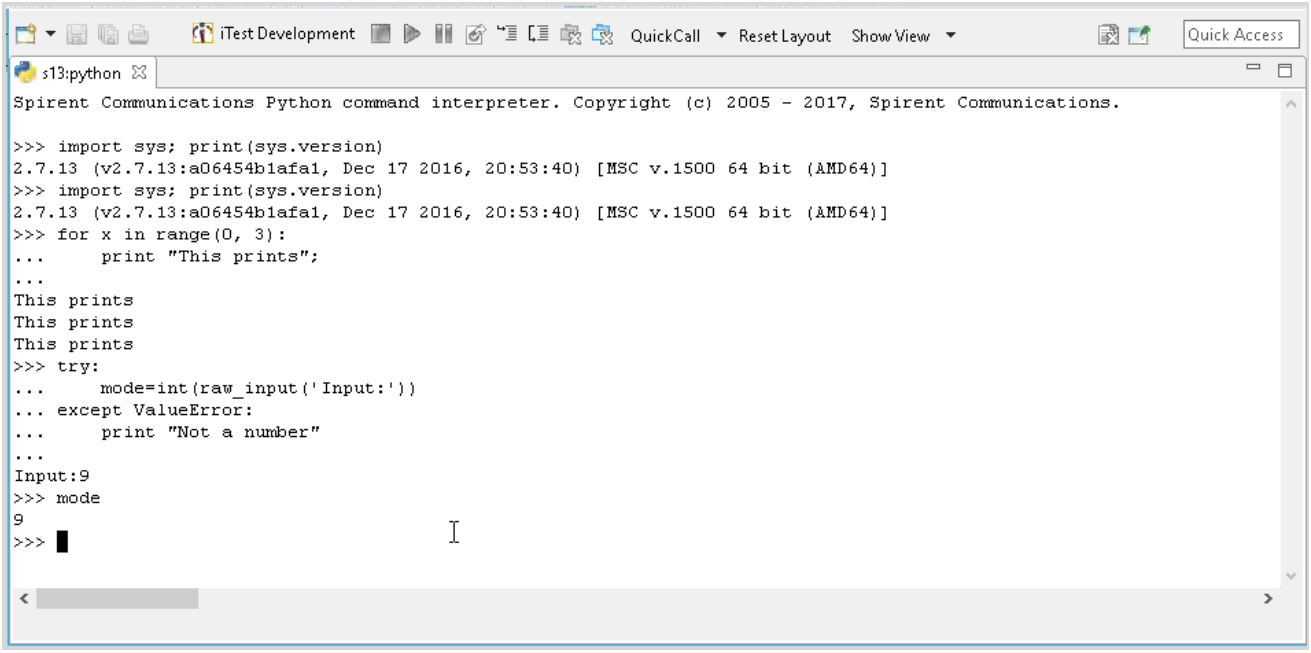
Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running Spirent iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Save session and Start the session. the Python terminal session opens.

Perform the required actions in the Python Terminal

iTest will start capturing the actions you perform. When you work in the iTest Python terminal session, iTest interprets the action and captures both the action and the response from terminal. See the example below.

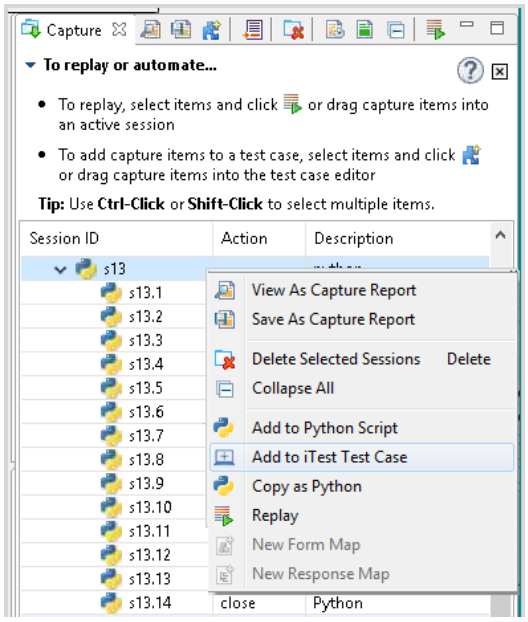


iTest captures the command and response on the Python terminal.

Note You may also run a script from the Python session.

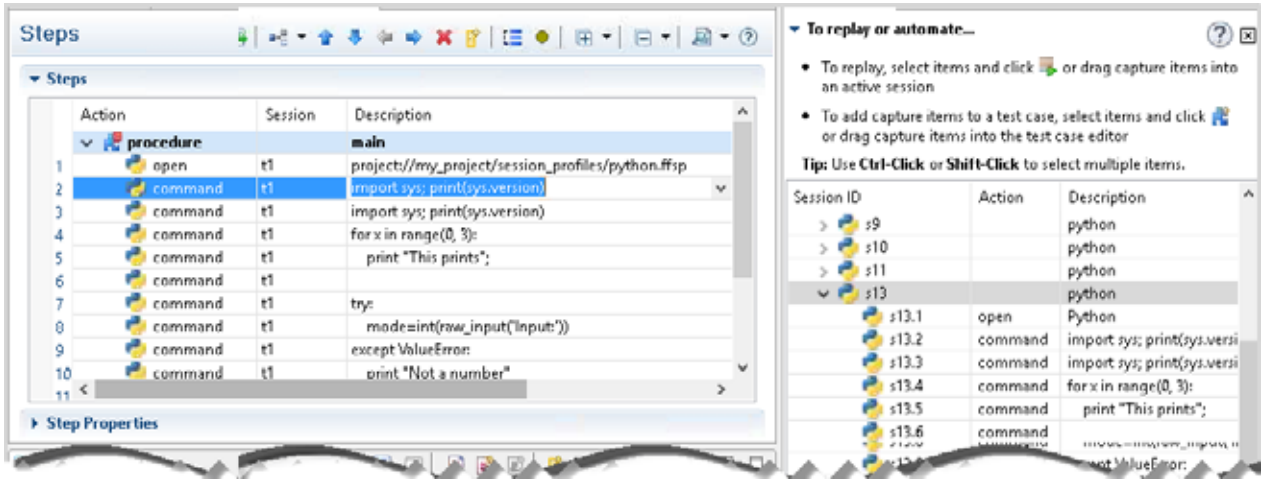
Add Python Actions steps captured to a Test Case

When you complete with the iTest Python terminal session, select the session and right-click to save the captured steps into a test case.



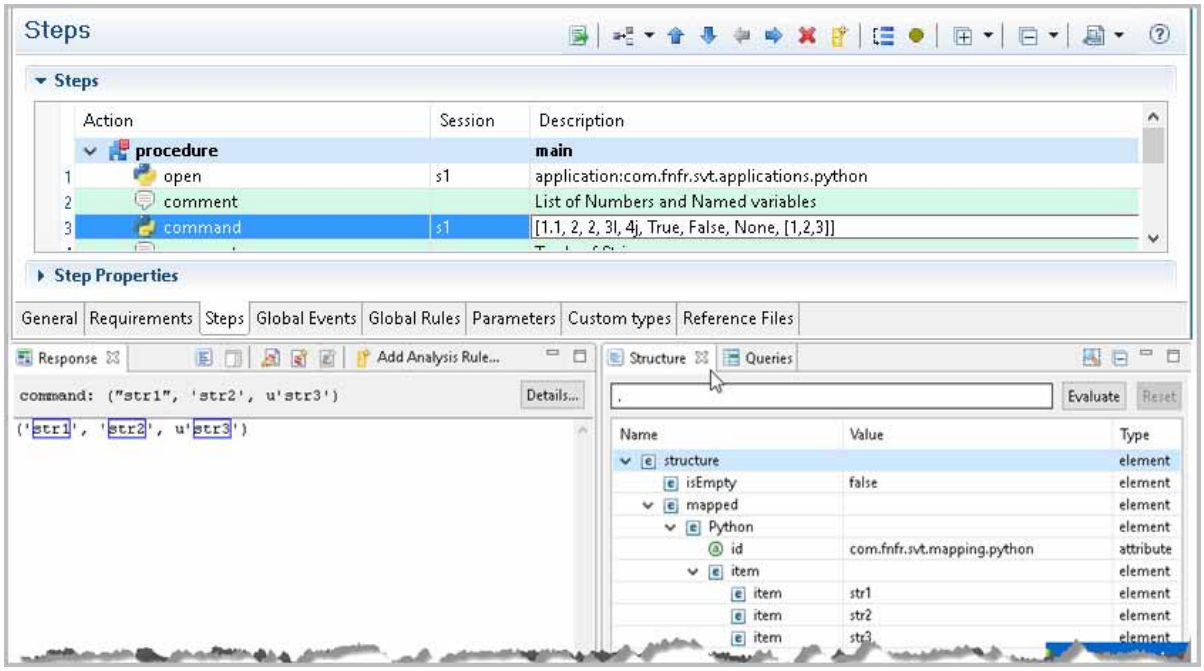
(The **Add Test Case** wizard opens and help you through the process. For details on saving captured steps to test cases, see "Saving interactive steps as steps in a test case: The 'Add to iTest Test Case' wizard" on page 98.)

The Test Case editor opens the new test case.



View Python Command steps Added to an iTest Test Case

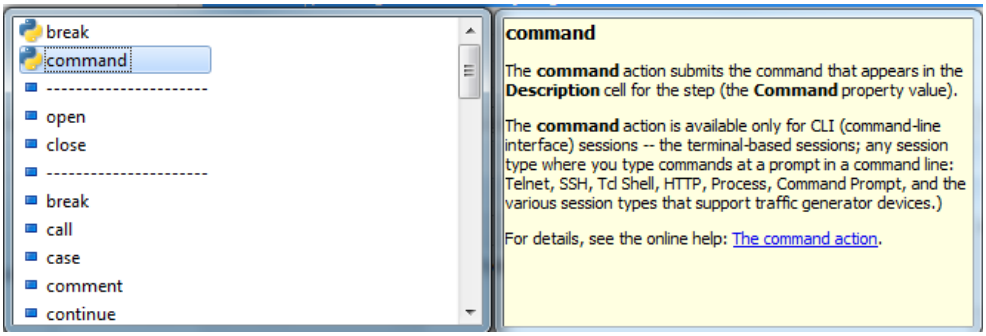
Select a command step and click the **Response** tab to see the repose map. That is, iTest auto-maps the responses when the captured steps are saved as a test case or added to an existing test case (that is, blue boxes appear in the **Response** view).



1. Continue editing the test case by adding analysis rules and flow-control steps as needed.
   - For information on working with test cases, see Chapter 7, “Test Cases”.
   - For information on using the Test Case editor, see Chapter 8, “Test Case Editor”.
2. In addition to the actions captured during the Python terminal session, you may add also add **ScriptSet** and **ScriptGet** commands (set and get the value of a variable) to the test case, if required.

   See Chapter 11, “Actions”, sections “The ‘scriptGet’ action: Get the value of a variable (Tcl or a selected interpreter)” on page 261 and “The ‘scriptSet’ action: Set the value of a variable (Tcl or a selected interpreter)” on page 262.

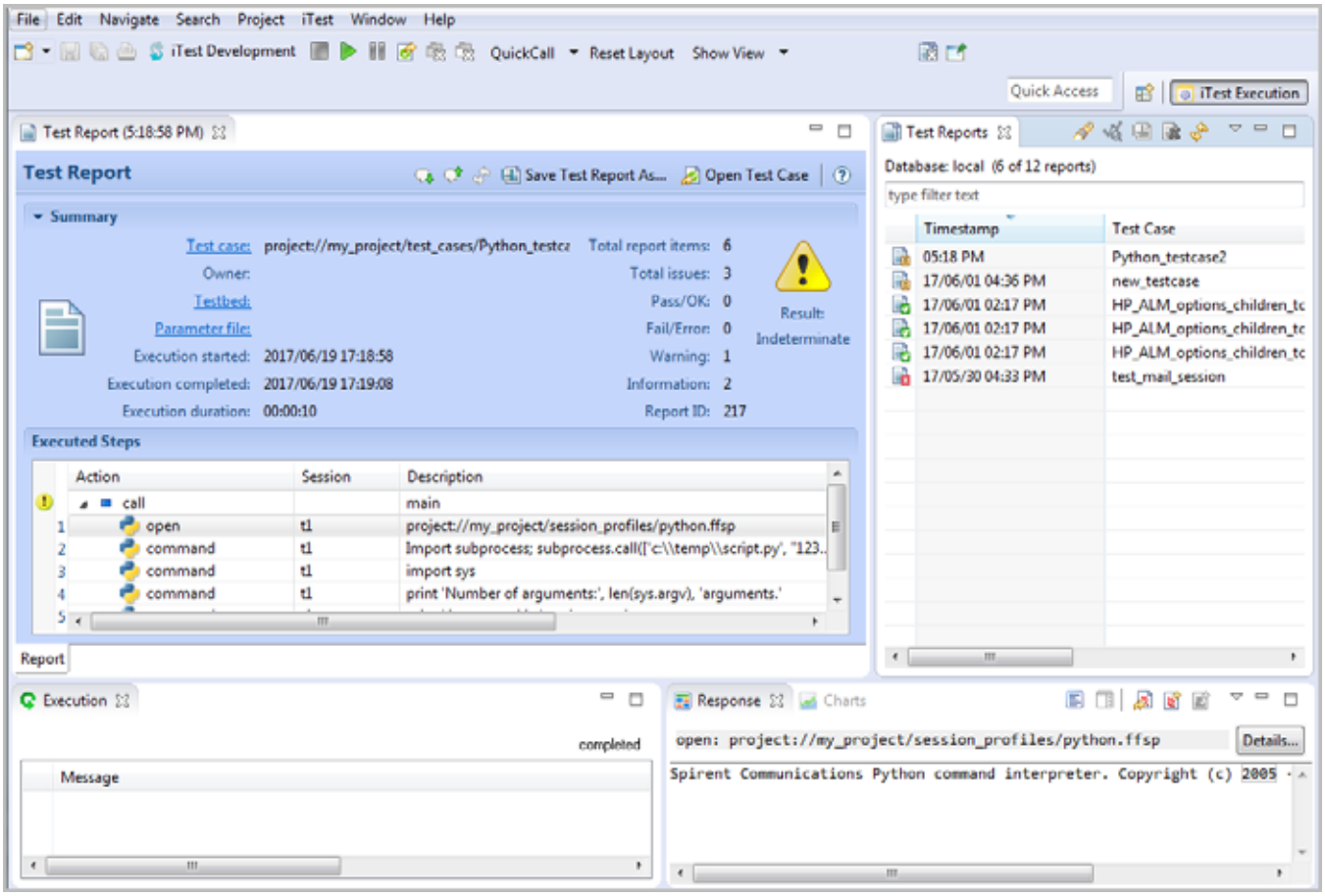
   To add a step in a iTest Python session test case, click in the **Action** cell (**ctrl + Space**) and select the action from the list of commands.



Execute Python session Test Case

You may run the Python command steps captured to a Test Case as any the other test case. When the execution completed, the test report will be generated.

Open the test case, click Start Execution in New Window. iTest Python terminal session opens and starts executing the steps in the test case. When the execution completed, the test report will be generated.



Setting preferences for Python

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Python**.

Configure iTest to auto-select the General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Python

Auto-select	Indicates that iTest will select the Python interpreter from the “Path” environment variable. **Note** This is the path used by all sessions, when no path is provided in the Session profile.
Use specified Python interpreter	Indicates that iTest will select the Python interpreter from the path specified, for example, **C:\Python36\python.exe** **Note** This is the path used by all sessions, when no path is specified in the Session profile.

CHAPTER 59

Python Automation Library

Overview

Spirent provides a Python Automation Library allowing step level interaction with iTest sessions. The library can be leveraged in your Python-based automation scripts and suites to drive commands and quick calls on multiple concurrent sessions. The library is a thin client for the following two iTest server instances:

- iTest GUI running on the local machine or on a remote host
- A Velocity agent running on the local machine or on a remote host.

The Python library controls iTest sessions on the (GUI or agent) and enables iTest services to be consumed within Python scripts, e.g., quick calls and response maps. See "Python Session Level Control Library" on page 1171. In addition, the iTest GUI can be used to generate example Python code from captured steps. See "Python Script Generation" on page 1191.

The following lists the use cases of Python Automation Library:

- To open existing session profiles (inside or outside a topology) so your Python script can:
  - Invoke quick calls with arguments.
  - Parse responses from quick calls (auto-mapped or explicitly-mapped).
- To issue native commands for CLI sessions so your Python script can:
  - Send any arbitrary command to an open session.
  - Parse responses from commands (auto-mapped or explicitly-mapped).
- To open and control built-in session independent of a pre-existing session profile.
- To use Python Automation Library commands with special step properties in an opened session profile.

Modes of Operation

The Python Automation Library provides these modes of Operation, connectivity, license usage.

- Connect to an iTest GUI instance ("iTest GUI Mode" on page 1168)
  - On the local host or remote host
  - Requires that Session Level Control agent be enabled and in listening mode (see "Configure Listening Mode (Session Level Control Agent)" on page 1169)
  - iTest GUI can use either Enterprise or Runtime license

- Connect to an agent (running agent instance) on a remote workstation ("Remote Mode" on page 1168).
  - On the local host or remote host
  - Requires that the agent is already started and in "listening" mode for SLC connections (see "Configure Listening Mode (Session Level Control Agent)" on page 1169)
  - Agent requires a Runtime license at startup
- Connect to a standalone workstation ("Standalone Mode" on page 1168)
  - On the local host
  - Auto-launch an agent in the background
  - Agent requires a Runtime license at startup

In the first two cases, iTest opens a socket and listens for connections. The listening mode is enabled via a specific agent command line parameter "–listeningMode", and on iTest GUI via "Configure Listening Mode (Session Level Control Agent)".

iTest GUI Mode

You may configure the listening mode for the iTest GUI via the Session Level Control Agent option (see "Configure Listening Mode (Session Level Control Agent)" on page 1169). If enabled, iTest GUI opens the listening socket at startup (or when enabled). The agent can either be connected to Velocity or available as a session level control agent, but not both at the same time.

Note iTest GUI becomes the agent running in listening mode. That is, the Python library can act as a client, controlling iTest at a step level.

In the iTest GUI mode, the Python Automation Library connects to the iTest GUI. If already connected (e.g., another instance of the library on a remote workstain), a new connection attempt would fail. Thus, only one Spirent Python Automation Library instance can use an iTest GUI instance at a time.

Remote Mode

The Velocity agent can be started manually in 'listening" mode to serve local and remote Python Automation Library clients. See "Remote Velocity Agent" on page 1174.

Standalone Mode

In the standalone mode, the library controls the agent startup process. During initialization, the library starts the agent in the background and connects to it. The agent is shut down when the library disconnects. If a separate Python script (separate process) tries to auto-launch its own agent in standalone mode, it is able to create and manage its own independent instance (process) of the agent.

Note Standalone mode, where the library auto-launches the agent, is the only case where the agent is shut down when the library disconnects. When the Python Automation Library disconnects from a GUI instance or an existing agent in listening mode, the GUI and agent instances remain up and are available for subsequent incoming connections.

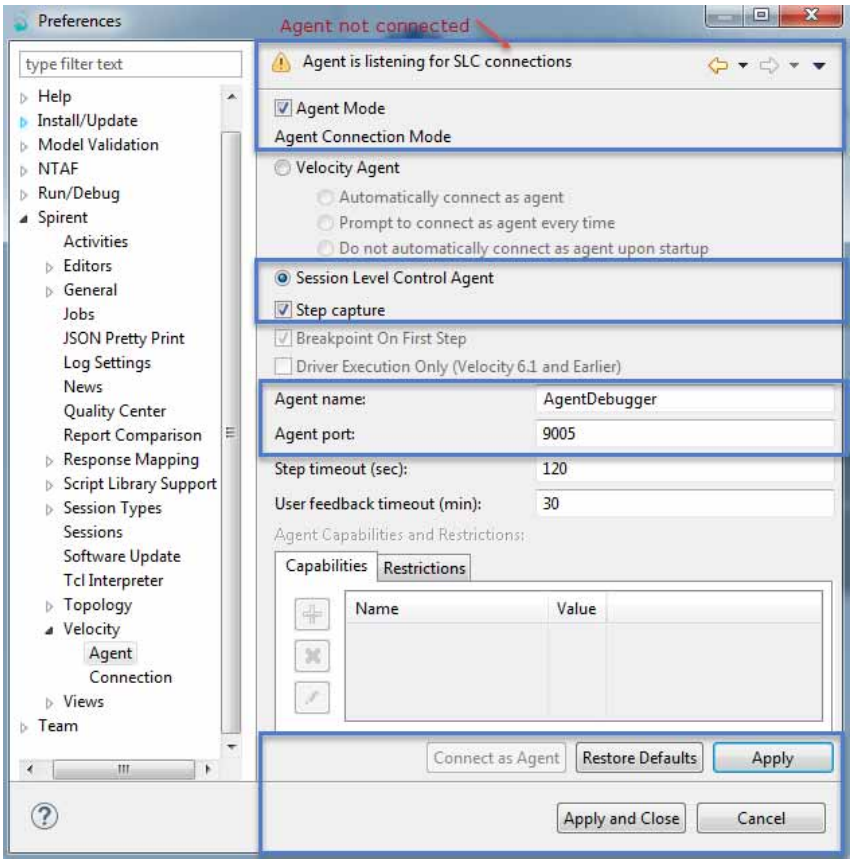
Configure Listening Mode (Session Level Control Agent)

Click **Windows> Preferencces> Spirent > Velocity > Agent**: enable **Agent mode** and complete as described below to enable **Session Level Control (SLC) Agent** mode. The following set up enables iTest to operate as an agent in listening mode:

- Configure Velocity Agent to run either as a Velocity **client** agent or an SLC **server** agent.
- Configure TCP listener port for incoming SLC agent connections.
- Configure Capture (optional **Step Capture**) mode for quick calls and commands issued from the Python API..

Agent Mode	Select to enable Agent Mode
Agent Connection Mode	N/A in SLC mode.
Session Level Control (SLC) Agent	This option enables the listening mode. iTest GUI will wait for client connections and does not connect to Velocity as an Agent. The **Velocity Agent Mode** is disabled when **Session Level Control Agent** is selected. iTest either connects to Velocity as an agent or acts as a **Session Level Agent** server. In **Session Level Agent** mode, the agent listens for an SLC connection (iTest GUI waits for connections) and the Python Automation Library connects to iTest GUI when available. **Note** Communication with the agent goes by sending and receiving Protobuf-serialized objects over TCP.
Step capture	The Step capture option is available only when the **Session Level Control Agent** is selected. • Selected: (Default), captures session actions (quick calls and commands) performed from the **Python Automation Library** on this iTest GUI instance. • Not selected: the session actions performed from the **Python Automation Library** library on this iTest GUI instance are *not* captured.
Breakpoint at first step	N/A in SLC mode. See "Configuring iTest Gui as an Agent" on page 1263 (Chapter 64, "Debug Velocity Drivers and Executions").
Driver execution only	N/A in SLC mode. See "Configuring iTest Gui as an Agent" on page 1263 (Chapter 64, "Debug Velocity Drivers and Executions").
Agent Name:	Enter a name for the Agent.
Port	Indicates the port used by the Agent during execution. • **Port for 6.1 (and earlier) Driver Agent:** Default: 9001 • **Port for Test Agent:** Default: 9000 • **Port for Session Level Control Agent**: Default 9005
Step timeout (sec)	N/A in SLC mode. See "Configuring iTest Gui as an Agent" on page 1263 (Chapter 64, "Debug Velocity Drivers and Executions").
User feedback timeout (min)	N/A in SLC mode. See "Configuring iTest Gui as an Agent" on page 1263 (Chapter 64, "Debug Velocity Drivers and Executions").
Agent Capabilities and Restrictions	N/A in SLC Mode. See "Configuring iTest Gui as an Agent" on page 1263 (Chapter 64, "Debug Velocity Drivers and Executions").

Apply and Close	Click to apply settings and connect as **Session Level Control Agent** and close the window. The Preferences window displays the connection state message depending on whether the Agent is connected or not as follows. When not connected: **Agent is listening for SLC conenctions** When connected: **Agent is connected**.
Restore Defaults **Apply**	**Restore default:** Click to discard all the changes made and reset to the default values. **Apply:** Click to apply the changes made.



Note See Chapter 41, “Configuring iTest Preferences” for general information on preference settings.

Python Session Level Control Library

Overview

This section describes installing, initializing (setting up), and using the Python Automation Library.

Installing/Upgrading Python Automation Library

Intall the Python Automation Library via pip:

```
pip install SpirentSLC
```

OR

```
easy_install SpirentSLC
```

Update the Python Automation Library via pip:

```
pip install SpirentSLC --upgrade
```

The Velocity Agent is not bundled with the Python Automation Library, and it should be installed separately. When a new version of the agent is released, users should update their Python Automation Library installation

Initializing/Setting up the Python Automation Library

The following initialization set up is required for Automation Library in Standalone mode and iTest GUI modes.

Standalone mode

Python API can auto-launch an agent on the local host, run in the background for the duration of the Python session (on port 9002 by default), and uses these (below) environment variables. Ensure that the following environment variables are set on the workstation where the library is installed and on which the script will run:

```
SPIRENT_SLC_HOST=local
# local is special keyword to startup Velocity Agent locally.
ITAR_PATH=path to folder where iTars are placed, can contain compressed or
exploded projects
SPIRENT_SLC_AGENT_PATH=a path to folder where iTest agent is located.
SPIRENT_SLC_LICENSE_SERVER=a license server
SPIRENT_SLC_LICENSE_PROXY=a license proxy for 64-bit agent.
```

ITAR_PATH set on the local environment indicates the folder where the iTars and exploded project folders are placed, so that the local execution agent can find the projects.

ITAR_PATH can be set as an environment variable on agent machine or specified as a command line argument.

The **ITAR_PATH** is not mandatory when connecting to a running instance of iTest GUI.

```
from SpirentSLC import SLC

slc = SLC.init()
```

Calling **SLC.init()** will initialize the underlying execution agent as a background process with which the library will communicate. An object is returned which is the entry point for further communication with the library.

Note The current release supports only one **init()** call within one Python interpreter context. An exception displays if unable to initialize the library. Every additional call will return same session object. For example:

```
>> slc1 = SLC.init()
>> slc2 = SLC.init()
>> slc3 = SLC.init()
```

The **SLC.init()** method also accepts two optional parameters to set license server information with the same meaning:

```
license_server
license_proxy
```

Example usage:

```
slc = SLC.init(license_server='testlshost.spirenteng.com:27000',
license_proxy='testlshost.spirenteng.com:5000')
```

Startup issues

If you encounter standalone agent startup issues, enable the verbose mode parameter and review the log.

```
SPIRENT_SLC_VERBOSE=True
```

Example of velocity agent startup output.

```
--->Agent verbose output
Starting velocity agent:
PATH: /Users/user/exaample/spirent/velocity-agent.app/Contents/Eclipse/
LOG FILE:
/var/folders/_m/dd5mtytd1zx9x8dx7vh7v7w40000gn/T/tmpzwzvd30xvelocity_agent
/agent.log
ITARPATH:
/Users/andrey/Develop/git/spirent/itest/dev/src/non-plugins/SpirentSLC/exa
mples/itars/
ARGUMENTS:
   --agentVelocityHost
   localhost
   --sfAgentServerPort
   9002
   --listeningMode
   library
   --sfAgentDisableSslValidation
   --licenseServer
   itest-lic
   --licenseProxy
   itest-lic
   --itar
/Users/user/example/git/spirent/itest/dev/src/non-plugins/SpirentSLC/examp
les/itars/
--->
```

Note The log folder and log file will be automatically deleted after the session is closed and all resources used by the Python Automation Library is released sucessfully.

```
slc.close()
```

iTest GUI mode

Ensure that the following environment variables are set on the workstation where the Python Automation Library is installed and the script will run:

```
SPIRENT_SLC_HOST=localhost:port  # must be host and port of the configured
instance of iTest GUI
```

For example, if the GUI is configured with defaults, the environment variable would be set to:

```
SPIRENT_SLC_HOST=localhost:9005
```

An instance of iTest must be running on the specified host and must be configured to accept connections at the desired port.

```
from SpirentSLC import SLC

slc = SLC.init() # will take all values from environment variables

# alternatively values may be provided in the init() call:

slc = SLC.init(host='localhost:9005')
```

An exception will display if the library is unable to connect to the iTest GUI instance.

Remote Velocity Agent

The Velocity agent can be started manually in ‘listening" mode to serve local and remote Python Automation Library clients.

Follow these steps.

1 Start a Velocity agent

```
./velocity-agent --agentVelocityHost localhost --sfAgentServerPort 9005
--listeningMode --licenseServer mylic-server --licenseProxy mylic-proxy
```

Note The host and the port must be the same as those specified in the environment variables.

2 Set the following environment variable from the Python Automation Library host machine:

```
SPIRENT_SLC_HOST=myhost:myport  # a host and port of velocity agent
```

3 Initialize SLC:

```
from SpirentSLC import SLC

slc = SLC.init() # will take all values from environment variables
# alternatively values may be provided in the init() call:
slc = SLC.init(host='localhost:9005')
```

Tip The host and the port must be the same as those specified in the environment variables.

Usage examples:

```
set ITAR_PATH=z:\downloads\itar

velocity-agent.bat --agentVelocityHost localhost --sfAgentServerPort 9002
--listeningMode --licenseServer 10.100.86.1 --itar z:\downloads\itar
```

OR

```
velocity-agent.bat --agentVelocityHost localhost --sfAgentServerPort 9002
--listeningMode --licenseServer 10.100.86.1
```

Note Ensure that you escape (backslash (\ ) character) all strings with backslash (\).

- Ensure escape (backslash (\ ) character) ITAR_PATH path when setting up the path from python script. For example:

```
if __name__ == '__main__':

    with SLC.init(itar_path='C:\\Users\\spirent\\Desktop\\testing',
    license_server='velocity-testlshost.spirenteng.com') as slc:
    main(slc)
```

See https://docs.python.org/2.0/ref/strings.html for more information about working with string in Python.

Working With Projects

Once initialized, the library will have access to all projects available in the iTest workspace (GUI) or itar path (Velocity agent). Each project contains a number of entities that can be addressed using the python code. These include session profiles and topologies.

Listing Projects

```
slc.list()==> ['project1_name', 'project2_name', 'project3_name']
```

All spaces in the name of a project or any other characters that are not legal in a Python identifier will be replaced by underscores in the returned values.

Importing/Opening Projects

To access session profiles and topologies in a project, the project must first be “imported” or “opened”. Since **import** is a reserved word in Python, it is called **open** in the **Spirent Python Automation Library.** Use the following code to import a project:

```
proj = slc.open('project1_name')
projs = slc.project1_name.open() # Alternative for the same method.

# multiple projects can be imported if needed

proj2 = slc.open('project2_name')
proj3 = slc.open('project3_name')
```

Querying a Project

```
# list all the usable topologies and session profiles in the project

proj.list()
==> ['dut1_ffsp', 'lab1_setup_tbml']
# list other types of assets, such as parameter files and response maps
proj.list(parameter_file=True, response_map=True)
==> ['dut1_ffsp', 'lab1_setup_tbml', 'main_setup_ffpt',
'response_map1_ffrm']
# show all QuickCalls available on a given session profile
proj.dut1_ffsp.list()
==> {
 'init_routes': {
  'all': 'True if all routes should be initialized'
 },
 'do_something_cool': {
  'param': 'Description of parameter'
 }
}
# access help on QuickCalls on a session attached to a resource in a topology
proj.lab1_setup_tbml.router1.ssh.list()
==> { ... same as above }
# access the list of parameters for a specific QuickCall
proj.dut1_ffsp.list('init_routes')
==> { 'all': 'True if all routes should be initialized' }
```

Built in session actions are not listed, only the QuickCalls attached to the session profile. If you are accessing a built-in session type such as Telnet or SSH, they may still invoke the actions, but they will not be listed by the **list()** call.

Note All displayed QuickCall names will be transformed into snake-case to conform to Python naming conventions.

Working with Sessions

You may invoke python commands (steps) in an open session using custom step properties passed as arguments to the step.

- The actions invoked interactively (e.g., REST POST steps, Selenium steps, STC steps) can also be invoked from the Python Automation Library
- The Python script generator ("Python Script Generation" on page 1191) also formulates the appropriate Python code related to step properties.

Note The Step properties conforms to a logical mapping from XML step properties to their corresponding properties in Python syntax.

The Python Automation Library also allows you to open a session without an associated session profile or topology file. See "Opening a Python built-in Session Type" on page 1183 for the complete list.

Basic Python Automation Library Commands

Basic Commands	Description
SLC.init()	Connects to an iTest instance (GUI or agent)
slc.list()	Displays the available projects in the SLC connection (where slc is the SpirentSLC connection object)
slc.open()	Imports a project to use during the SLC connection (where slc is the SpirentSLC connection object)
project.sessionProfile.open()	Opens a session within an opened project (where project is the opened project object and sessionProfile is a standalone session profile or one in a topology)
session.quickCall()	Invokes a quick call available in the opened session (where session is the opened session object and quickCall is an availble quick call associated with that session)
session.command (‘myCommand")	Issues a command within an opened CLI session (where session is the opened session object and myCommand is the command sent to the session)
response.text()	Displays the text response from a quick call or command from an active session (where response is the command response object)
response.queries()	Lists the available queries (auto-mapped or explicitly mapped) from the response of a quick call or command in an active session (where response is the command response object)
response.json	Displays the dictionary object associated with a JSON response (where response is the command response object)
slc.sessions.sessionType. open()	Opens a ‘built-in" session in the SLC connection (where slc is the SpirentSLC connection object)
builtInSession.command (‘myCommand")	Issues a command in an opened built-in session (where builtInSession is the opened session object and myCommand is the command sent to the session)
session.action (‘argument", properties={})	Invokes a built-in action using custom step properties (where session is the opened session object, argument is the general argument, and properties contains a structure defining custom step properties)
session.session_properties()	Lists the available session property syntax that can be used to override settings on a session open operation (where session is the opened session object)
session.step_properties (“action”)	Lists the available step property syntax that can be used to override default settings on a step operation (where session is the opened session object and action is a built-in action on the session)

Opening a Session

Sessions are opened either directly on a session profile or local topology.

```
# open session on a session profile
s1 = proj.dut1_ffsp.open()

# open session, giving required parameters
s1 = proj.rest_session_ffsp.open(url='https://my_site.my_domain.com',
accept_all_cookies=True)

# open session, using parameter file
s1 = proj.rest_session_ffsp.open(parameter_file=proj.main_setup_ffpt)

# open session, specify a reponse map library to use
#response_map_library must be open if you want to specify a response map library when
opening a session in Python console
resp_lib=slc.response_map_library.open()

s1 = proj.rest_session_ffsp.open(response_map_lib=resp_lib)

# resp_lib could be replaced with project notation instead, for example:
‘project://response_maps’

# open session, specifying additional session properties
s1 = proj.rest_session_ffsp.open(properties={'authentication.authentication': 'Basic',
'authentication.user': 'me', 'authentication.password': 'totes_secret!'})

# open session on a resource in a local topology
s1 = proj.lab1_setup_tbml.router1.ssh.open(...) # may use any combination of parameters,
parameter_file, agent_requirements, properties

# Previously opened sessions could be accessible using:
# dut_project.<session_name>_ffsp and could be # assigned to any other variable:
#Example:
# s1=dut_project.Cisco3750_ffsp.open()
# .. some actions
# s2=dut_project.Cisco3750_ffsp
```

Queries Session properties

You may query for Session Properties available after a session is opened.

```
# open session, specifying additional session properties
s1 = proj.rest_session_ffsp.open(properties={"authentication.authenticationType":
"Basic", "authentication.user": "me", "authentication.password": "totes_secret!"})
# Query for session properties
s1.session_properties()
List of session properties: com.fnfr.itest.applications.webservices.restful
"url": "https://jsonplaceholder.typicode.com/"
"acceptAllCookies": "false"
"autoRedirect": "true"
"authentication.authenticationType": "Basic"
"authentication.user": "me"
"authentication.password": "totes_secret!"
"authentication.keyStoreFile": ""
"authentication.passphrase": ""
"authentication.acceptAllCertificates": "true"
"httpHeader": "[]"
"proxy.useProxy": "false"
"proxy.proxyHostName": "localhost"
"proxy.proxyPort": "8888"
"proxy.userName": ""
"proxy.password": ""
"urlParameters": "[]"
```

Note Each individual session has different set of properties available and may be specified during session open.

Handling session open status

To handle session opening status we need to check a response object.

```
# Session open response will be stored into
s1.open_response -> A Session Action Response object with output, status etc.
```

The above open_response object is same as any response object described in “Response” on page 1181.

Session Information

Once a session is opened it is possible to find out some basic information about where the session is being handled. This is done via the agent property of a session object.

```
# Use the print command when using a standalone agent
>>> print(s1.agent)
{'agent_name': u'USER01-PC', 'agent_type': 'local', 'name': u'USER01-PC',
'capabilities': {u'Product.Arch': u'x86', u'OS.Type': u'win32',
u'STC.Version': u'4.69', u'language': u'itest'}, 'protocol_version': u'1.0'}
```

Invoking Actions on Session

An active session has a number of actions associated, which may be either built-in actions or QuickCalls defined on that session type. Any of those can be invoked on the session.

```
# invoke the init_routes QuickCall with one parameter
response = s1.init_routes(all="True")

# invoke a built-in action with a specific response map (which may override what
was set for the session as a whole)
response = my_ssh_session.command('ls',
response_map="proj.response_map_ls_ffrm")
```

Execute step with properties

Example 1:

```
ssh = slc.sessions.ssh.open(properties = {
    'ipAddress':'velocity-ileana.spirenteng.com',
    'user':'root',
    'password':'iRyLc4KQj80=',
    'TerminalProperties':{
       'prompts': [ { 'prompt1':{ 'Content':'root@velocity-ileana:~#' } }, {
       'prompt2':{   'Content':'root@velocity-itleana:/#' } } ]
     }
    })
```

Tip Since only encoded passwords are supported, it is recommended (the easiest way) to create session profile with SSH session and copy the encoded password from fftc file.

When executing native sessions in Python console, to receive an entire command response, add More prompt as a paramter.

```
'prompt':{ 'Content':'--More--', 'TypeOfPrompt':'MORE' }
```

```
>>>telnet_session=slc.sessions.telnet.open(properties =
{ 'ipAddress':'10.140.x.x', 'TerminalProperties':
{'prompts': [{ 'prompt1':{ 'Content':'3750>' },
'prompt2':{ 'Content':'3750#' }, 'prompt3':{ 'Content':'--More--',
'TypeOfPrompt':'MORE' },
'prompt4':{ 'Content':'Username:' }, 'prompt5':{ 'Content':'Password:' }}]}})
```

Example 2:

```
# open session, specifying additional session properties
```

```
>>> s1 = slc.sessions.rest.open(properties={ 'url':'http://example.com',
'httpHeader':[ { 'header':'H1', 'value':'H2' } ], 'httpHeaderTemplate':
[ { 'header':'Content-Type', 'value':'application/json' }, {
'header':'Cookie', 'value':'x' }, { 'header':'Content-Type',
'value':'application/json' }, { 'header':'Cookie', 'value':'x' } ] })
```

Example 3:

```
#Execute step, specifying additional step properties
```

```
>>> s1.GET('http://example.com/index.html', properties={
'applicationProperties.action':'http://example.com/index.html',
'applicationProperties.formData':[ { 'Name':'Abc', 'Value':'Def' } ] })
#Alternative using the dotted notation
>>> s1.GET(properties={"qualityCenterStepInfo.stepName": "",
"qualityCenterStepInfo.associateWithDesignStep": "false"})
```

Response

The resulting response object can be used to query details about the action execution and the response itself:

```
# duration of execution and any error status
>>>response.duration
3 # number of seconds
>>>response.result
'success' # may be success, failed, timeout
# textual rendering of the response
>>>response.text
'textual response data'
# if the response is json, it is easier to grab the json directly as a dictionary
>>>response.json
instance of dictionary # null if not available as json

# if the response is xml, memory location is printed in Python console when trying
to access a http response in XML format
>>>response_http.xml
<Element 'Xml' at 0x062603F0>

>>>response.data
{ a structured data object with step structured data }
#Example of structure data object from 'command' session:
>>> response.data
[map: "*"
 items {
 name: "isEmpty"
 value: "false" }
 items {
 name: "promptName" value: "defaultPrompt"
 } items {
 name: "echo"
value: "dir" }
items {
name: "prompt"
value: "C:\\Windows\\system32>"
} ]
```

Queries

The response object may also have queries defined on it - methods that query the structured data and return values. Queries may be auto-generated in iTest or be defined in response maps.

```
# list the set of queries that exist for the response

response.queries()
==> [ 'is_empty()', 'counter_by_row(row)' ]
# invoke query
response.counter_by_row(3)
==> 35
```

Note Query names are always converted to snake case.

Checking available step properties

You may query a session for step properties command after the session is opened.

Any of these properties may be passed to the Steps properties. iTest maps the python script to text representation of .fftc files.

Note The following is an example query and the output differs for every iTest session.

```
# open session, specifying additional session properties
>>> s2.session_properties()
List of session properties: com.fnfr.itest.applications.webservices.restful
"url": "http://spirent.com"     # string value
"acceptAllCookies": "false"    # boolean value
"autoRedirect": "true"  # boolean value
"authentication.authenticationType": "None" enum value one of [None, Basic,
Secure]
"authentication.user": ""     # string value
"authentication.password": ""  # string value
"authentication.keyStoreFile": ""     # string value
"authentication.passphrase": "" # string value
"authentication.acceptAllCertificates": "false" # boolean value
#... And other properties.
# query for step properties.
>>> s2.step_properties('GET')
List of session properties: com.fnfr.itest.applications.webservices.restful
"guid": "9284bfbe-9dc5-443c-b4f0-f5af4af87dca"  # string value
"action": "GET" # string value
"session": "0555bbba-c858-11e7-8c32-56003d995201"     # string value
"context": ""  # string value
"target": ""   # string value
"async": "false"      # boolean value
"isBackgroundThread": "false"  # boolean value
"threadName": ""      # string value
"skip": "false" # boolean value
"normalOffset": "0s" time-span value
"acceleratedOffset": "0s" time-span value
"estimatedStepExecutionTime": "0s" time-span value

"command.headers": "{}"
"command.contentType": "TEXT" enum value one of [TEXT, XML]
"command.body": "None"  # string value
"command.isEncrypted": "false"  # boolean value

"documentation.label": ""     # string value
"documentation.tag": "" # string value
"documentation.comment": ""    # string value

...

...
```

Closing a Session

Sessions should be closed when no longer needed, as they consume resources on the agent (and on Velocity if being used.) It is especially important to close sessions if sessions are being opened within a loop.

```
# close session and free resources

s1.close()
```

Shutdown

Proper shutdown of the library is important to ensure timely release of resources.

```
# release all resources used by the library

slc.close()
```

Resources released include all remaining open sessions, all reservations initiated by the script, and (if local) the underlying execution agent.

Opening a Python built-in Session Type

You may open a session using the Python Automation Library without having an underlying session profile or topology file.

```
s1 = slc.sessions.ssh.open()
#open the built-in ssh session type directly, supplying required session profile information
ssh = slc.sessions.ssh.open(properties = {
'ipAddress':'velocity-itest3.spirent.com',
'user':'root',
'password':'iSyGc4KGj80=',
'TerminalProperties':{
'prompts': [ { 'prompt1':{ 'Content':'root@velocity-itest3:~#' } }, { 'prompt2':{
'Content':'root@velocity-itest3:/#' } } ]
}
})

#The following is a list of available sessions.

>>> slc.sessions.agilent.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.agilent

>>> slc.sessions.avalanche.open()
#Return session profile obj with
URI=application://com.fnfr.svt.applications.spirent.avalanche

#There are several options for running an Avalanche test in iTest:
#* Running a test using the Tcl scripts generated by Avalanche
#* Running an Avalanche test on a TestCenter device
#* Running a test using Demo mode

>>> slc.sessions.chat.open()
#Return session profile obj with URI=application://com.fnfr.bobcat.tools.chat
#You can add steps that receive and send XMPP chat messages during execution. A test case can send
and #receive as many messages as are needed. Message text can contain both fixed text and
response data

>>> slc.sessions.cmd.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.cmd
#You can run Microsoft Windows Command Prompt sessions (that is, run cmd terminal sessions). For
each open session, #the Command Prompt Session window displays your commands and the local PC's
responses.
```

```
 >>> slc.sessions.database.open()
#Return session profile obj with URI=application://com.fnfr.itest.tools.database
#The Database Client session window is an interactive browser where you perform database operations and monitor responses.
#iTest captures all commands and responses and you can save captured items as test case steps that start the session and #set and request database records. Automated test cases open the same session window to perform database operations. #iTest supports sessions with MySQL, SqlServer, Oracle, Szlite, Derby, or a custom database type that you specify.


>>> slc.sessions.eggplant.open()
#Return session profile obj with URI=application://com.spirent.itest.applications.testplant
#EggPlant Test session provides ways of automating execution of image-based GUI testing. The seamless testing of these iTest functionalities #is achieved by iTest integration with EggPlant application (http://www.testplant.com/eggplant/testing-tools/eggplant-developer/).


>>> slc.sessions.file.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.file
#The File session type enables an automated test case to work with text files during execution (open a
#file, go to a specified position in the file, read a line, write a line, and so on).


>>> slc.sessions.flex.open()
#Return session profile obj with URI=application://com.fnfr.itest.tools.flex
#iTest supports testing Flash applications that were developed using Adobe Flex.
#iTest can capture your interactions with Flex applications that are hosted on web pages.
#iTest captures each step of an interactive (manual) test in the Flex application.


>>> slc.sessions.http.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.http
#Using an HTTP session, a test case can talk directly with a device using the HTTP protocol operations GET and POST.
#HTTP GET commands are useful in cases where you are not testing a Web application, but rather are testing something
#like a device through which the HTTP is passing.


>>> slc.sessions.ixiaTraffic.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.ixiaTraffic
#The Ixia Traffic session window is an interactive terminal where you enter commands to perform Ixia Traffic actions
#on the device. The session on the Ixia device returns text responses.
```

```
>>> slc.sessions.ixia_ixload.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.ixia.ixload
#To run an IxLoad session in iTest, you first define a test or tests in IxLoad and save the configuration file (RFX file)
#as you normally would. You then start the IxLoad session in iTest and run the IxLoad test. When the session starts, it checks
#out the libraries and connects and runs the initialization script and then loads the configuration, validates it, and transforms
#it (see the description for the iTest load command). The test produces responses and CSV files, which you can then post-process.
#The IxLoad session window is an interactive terminal where you enter commands to perform IxLoad actions on the device. IxLoad returns
#text responses. The responses to IxLoad responses are structured and iTest uses built-in mappers to supply read-made queries.


>>> slc.sessions.ixia_ixnetwork.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.ixia.ixnetwork
#In IxNetwork sessions, you can start and stop traffic, start and stop capture, and review statistics.
#The levels of the Object Data Matrix are represented as subdirectories (see Table 1-11, API Command Data
#ModelStructure in the IxNetwork Tcl API Guide). The iTest IxNetwork session window enables you to navigate
#the object hierarchy by using commands that you typically use to navigate a file system.


>>> slc.sessions.landslide.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.landslide


>>> slc.sessions.landsliderest.open()
#Return session profile obj with
URI=application://com.spirent.itest.applications.landsliderest
#You can use iTest to automate Spirent Landslide tests.
#In iTest, when you start a Landslide session, iTest launches the Landslide TAS user interface running on the Landslide device.
#1. Now you can interact with the TAS in the normal way. For example, you might load a test configuration, start the test session,
#collect the responses, wait to collect several data sets, stop the test session, request the test session results, and then close the test session.
#2. When you are finished working on the TAS, you return to iTest and close the Landslide session window.
#3. You can now save the captured steps and responses as an iTest test case. As needed, you can modify and update the test case (for example, modify
#the ImportTestSuite step by causing it to load a different Landslide test suite file or replace the filename with a variable whose value is set at runtime).
```

```
>>> slc.sessions.mail.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.mail
#You can add steps that construct and send email messages during execution. A test case can construct and send as
#many email messages as are needed. The message body can contain both fixed text and test response and result data.


>>> slc.sessions.netconf.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.spirent.netconf.tool
#The NetConf session window displays your commands and the device's responses. You can think of the session window
#as aterminal - a terminal to a NetConf service as a subsystem that iTest is monitoring and capturing.


>>> slc.sessions.popmail.open()
#Return session profile obj with URI=application://com.spirent.itest.applications.popmail
#See the Mail (POP3) sessions to retrieve emails from subscribers, view content, extract the required text and
#attachments, save the attached images as individual files, and then insert them into other sessions, for example, Selenium.


>>> slc.sessions.posapp.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.spirent.posapp.tool


>>> slc.sessions.powershell.open()
#Return session profile obj with URI=application://com.spirent.itest.applications.powershell
#PowerShell sessions execute commands at the Windows PowerShell prompt. The PowerShell session
#window displays your commands and the local PC's responses.


>>> slc.sessions.process.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.process
#Execute and manage processes on the computer that is running iTest.


>>> slc.sessions.python.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.python
#iTest Python session is a terminal session, similar to Tcl Shell. The session uses native Python interpreter
#to getting responses. Supported versions of interpreters are: 2.4-2.7 and 3.0-3.6. iTest support an internal
#Python interpreter and also allows you to point to an external interpreter via Preferences settings.
```

```
>>> slc.sessions.ranorex.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.ranorex
#Ranorex Test session provides ways of automating the Windows, Web, and Mobile UI applications. The seamless
#testing of these iTest functionalities is achieved by iTest integration with Ranorex application (www.ranorex.com).


>>> slc.sessions.rest.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.webservices.restful
#The REST session window provides a work surface for composing and submitting HTTP requests. Any request that
#you submit in the iTest session is forwarded to the RESTful service. The service performs the action and returns
#its normal response. iTest captures all of the actions that you perform in a session and all of the responses.


>>> slc.sessions.rft.open()
#Return session profile obj with URI=application://com.fanfare.itest.appstore.applications.rft
#The session window for iTest RFT sessions has been designed in partnership with IBM to enable you to launch Rational
#Functional Tester scripts from iTest. This integration is certified by IBM as Ready for IBM Rational Software.


>>> slc.sessions.selenium.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.selenium
#For Selenium sessions, iTest opens an instance of the Firefox browser. You interact with the pages in the normal
#way while iTest captures your actions and responses from the session.


>>> slc.sessions.serial.open()
#Return session profile obj with URI=application://com.fnfr.itest.application.serial
#For Serial Port sessions, the computer running iTest communicates directly over a serial port connection with the device
# To increase the wait time (timeout) for serial session response
serial_session = dut_project.command('show?', properties={"applicationProperties.completionProperties.promptLearnUserResponseTime": "150"})
#under test. For each open session, the Serial Port session window displays your commands and the device's responses. You
#can think of the session window as a terminal client - a terminal that iTest is monitoring and capturing.


>>> slc.sessions.smartbits.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.smartbits
#The SmartBits session window is an interactive terminal where you enter commands to perform
#SmartBits actions on the device. SmartBits returns text responses.
```

```
>>> slc.sessions.snmp.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.snmp
#A hierarchical browser for getting and setting SNMP MIB data using the Simple Network Management Protocol
#(SNMP V1, V2c, or V3) defined in RFC 1157


>>> slc.sessions.spirenttestcenter.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.spirenttestcenter
#iTest integrates with Spirent TestCenter (STC) and provides REST API to ensure that the automation functionality is readily available
#to a wide variety of clients using both script and GUI. This integration also eliminates the requirement of an STC installation on the
#client system. This allows you to use existing tools available to create STC automation clients that can run on any platform.


>>> slc.sessions.telnet.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.telnet
#A virtual terminal that communicates with a device using the Telnet protocol defined in RFC 854
#Note: Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet
#or SSH session to "localhost" to access a shell.


>>> slc.sessions.testcenter_gui.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.spirent.testcenter.gui


>>> slc.sessions.udp.open()
#Return session profile obj with URI=application://com.fnfr.itest.applications.udp.tool
#For UDP sessions, the computer running iTest communicates directly over UDP (User Datagram Protocol) with the specified device.
#For each open session, the UDP session window displays your commands and the local echo. Open another session to view the device's
#response. You can think of the window as a terminal client - a terminal that iTest is monitoring and capturing.


>>> slc.sessions.ssh.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.ssh
#A virtual terminal that communicates with a device using the SSH protocol (SSH-1 or SSH-2) defined in RFC 4250
# Custom step properties when issuing ssh 'top' command
s1.command('top',
properties={'applicationProperties.completionProperties.completeWhen':'TIMED',
"applicationProperties.completionProperties.completionTime": "3",
"applicationProperties.completionProperties.screenMode": "true"})
s1.command('q') # to quit from console top command
#Note: Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet or SSH session
#to "localhost" to access a shell.
```

```
>>> slc.sessions.stcrest.open()
#Return session profile obj with URI=application://com.spirent.itest.applications.stcrest


>>> slc.sessions.syslog.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.syslog
#Each Syslog session monitors the syslog messages that arrive at the built-in iTest syslog server (visible in the Syslog view).
#While the syslog server receives all messages, any syslog session can filter the messages based on the following property settings
#in the session profile.
#As a result of configuring session profile settings, only the messages that meet the filter settings appear in the session window.
#This enables your test cases to analyze the particular responses (messages) of interest and to ignore irrelevant messages.


>>> slc.sessions.tclsh.open()
#Return session profile obj with URI=application://com.fnfr.svt.applications.tclsh
#You can type Tcl commands and expressions into the iTest Tcl Shell session window.
#iTest captures your commands and the interpreter's responses.


>>> slc.sessions.telnet.open()
# Return session profile obj with URI=application://com.fnfr.svt.applications.telnet
# A virtual terminal that communicates with a device using the Telnet protocol defined in RFC 854
# Note: Because Linux and Unix devices typically include Telnet and SSH servers, you can start a Telnet
# or SSH session to "localhost" to access a shell.


>>> slc.sessions.testcenter_gui.open()
# Return session profile obj with
URI=application://com.fnfr.svt.applications.spirent.testcenter.gui


>>> slc.sessions.tool.open()
# Return session profile obj with URI=application://com.fnfr.itest.application.sf.tool


>>> slc.sessions.udp.open()
# Return session profile obj with URI=application://com.fnfr.itest.applications.udp.tool
# For UDP sessions, the computer running iTest communicates directly over UDP (User Datagram Protocol) with the specified device.
# For each open session, the UDP session window displays your commands and the local echo. Open another session to view the device's
# response. You can think of the window as a terminal client - a terminal that iTest is monitoring and capturing.
```

```
>>> slc.sessions.vmware_core.open()
# Return session profile obj with
URI=application://com.fnfr.itest.applications.vmware.vi.core.tool


>>> slc.sessions.vmware_vscphere.open()
# Return session profile obj with
URI=application://com.fnfr.itest.applications.vmware.vsphere.core.too


>>> slc.sessions.vnc.open()
# Return session profile obj with URI=application://com.fnfr.itest.applications.vnc
# iTest VNC sessions are intended to help you to control a remote OS to perform
configuration/setup/tear-down tasks. iTest
# VNC sessions are not intended to enable you to thoroughly test an application on a remote
platform.


>>> slc.sessions.web.open()
# Return session profile obj with URI=application://com.fnfr.svt.applications.web


>>> slc.sessions.webservices.open()
# Return session profile obj with URI=application://com.fnfr.itest.applications.webservices
# The Web Services session window provides a work surface for composing and submitting Web
Service requests.
# Any request that you submit in the iTest session is forwarded to the Web Services server.
# The Web Service performs the action and returns its normal response.


>>> slc.sessions.windowsgui.open()
# Return session profile obj with URI=application://com.fnfr.itest.tools.windowsgui


>>> slc.sessions.wireshark.open()
# Return session profile obj with URI=application://com.fnfr.svt.applications.wireshark
# Wireshark sessions provide a command line interface for interactively capturing packets from
a network interface.
# For commands that return status and packet data, iTest saves the responses as structured data
and generates
# associated queries to simplify pass/fail analysis.


>>> slc.sessions.xmlrpc.open()
# Return session profile obj with
URI=application://com.fnfr.itest.applications.webservices.xmlrpc
# The XML-RPC session window provides a work surface for composing and submitting XML-RPC method
calls over HTTP and HTTPS.
# Any request that you submit in the iTest session is forwarded to the XML-RPC service. The
service performs the action and
# returns a response. You can view the response in the Response section of the XML-RPC session
window. iTest captures all of
# the actions that you perform in a session and all of the responses. You can use the captured
items to create test case
# steps that interact with the XML-RPC server.
```

Python Script Generation

Overview

The iTest GUI includes a utility that converts captured steps from interactive sessions to Python scripts. The exported code can then be used inside a larger Python test suite to produce highly customized test scripts written purely in Python. iTest generates Python scripts to help you quickly author your test cases using the Spirent Python Automation Library. See "Python Automation Library" on page 1167.

See "Generate/Copy Python Script from Captured Steps" on page 1192 for step by step instruction

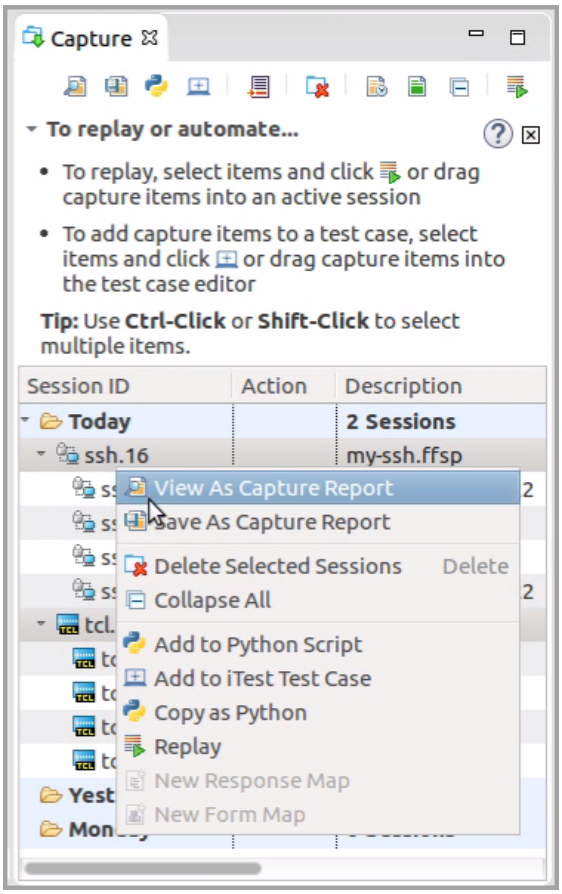
Before generating Python Script

Ensure the following are performed before attempting to generate Python Scripts:

- Install Python Automation Library. See "Python Session Level Control Library" on page 1171.
- Set up Session Level Control Agent. See "Configure Listening Mode (Session Level Control Agent)" on page 1169

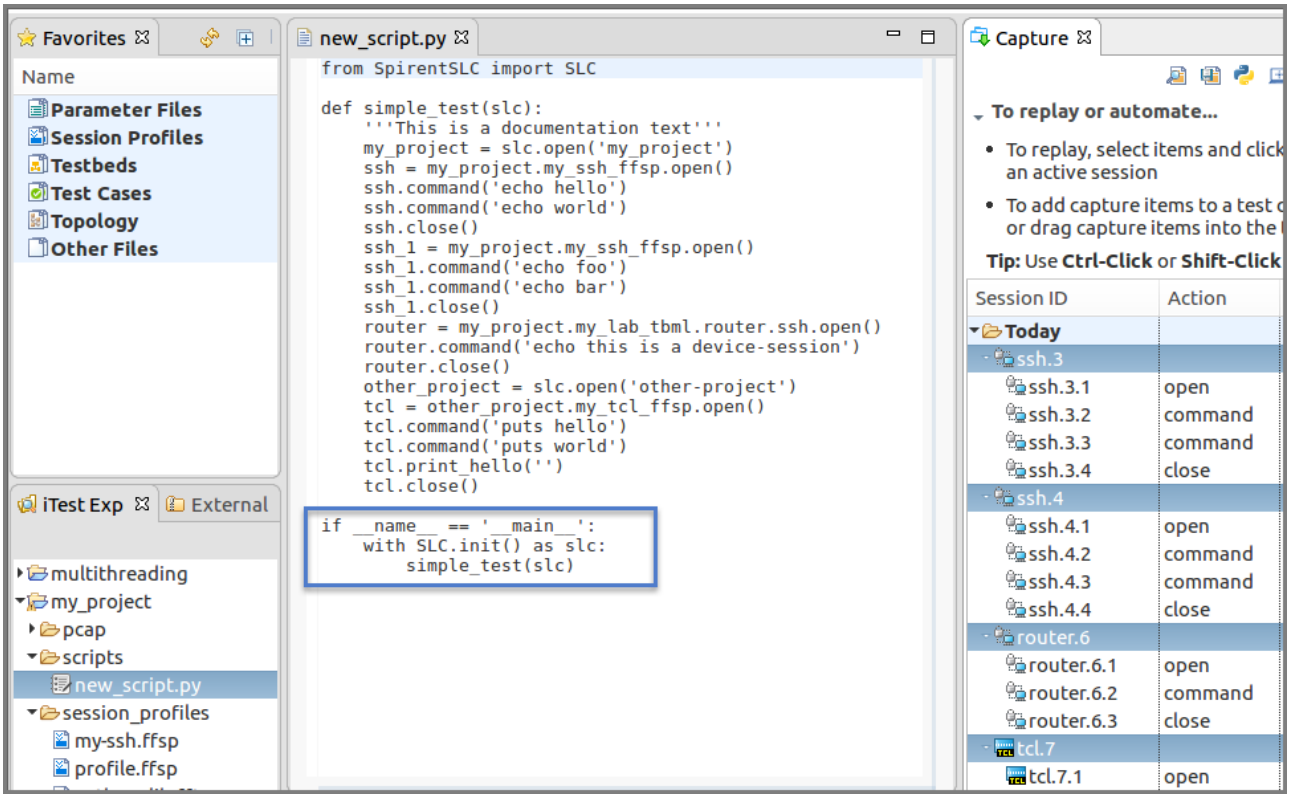
Generate/Copy Python Script from Captured Steps

You may select captured steps in the Capture View and then generate a Python script by clicking the **Add to Python Script** on the Capture View tool bar or via the right-click menu **Add to Python Script** or **Copy as Python**. iTest opens the Python Script in an embedded text editor.



Add to Python Script

Clicking this option on the Capture View toolbar or from the right-click menu, renders script that contains initialization code and all necessary import in addition to the steps.



- Includes the selected open steps and the import commands. Where applicable, includes the topology used.

  The test session object names used in the exported Python script uses the same session names shown in the capture view.

  Note You may specify a name and a documentation string of the procedure that will contain captured steps.

- The generated Python script will contain code for all selected captured steps, including the necessary import code for imports and library initialization, e.g., **from SpirentSLC import SLC**, and **init()** method to connect to the localhost iTest GUI.

  The Python Script includes custom step properties, that is,

  - step properties are rendered in the generated Python code
  - includes step properties in the session commands

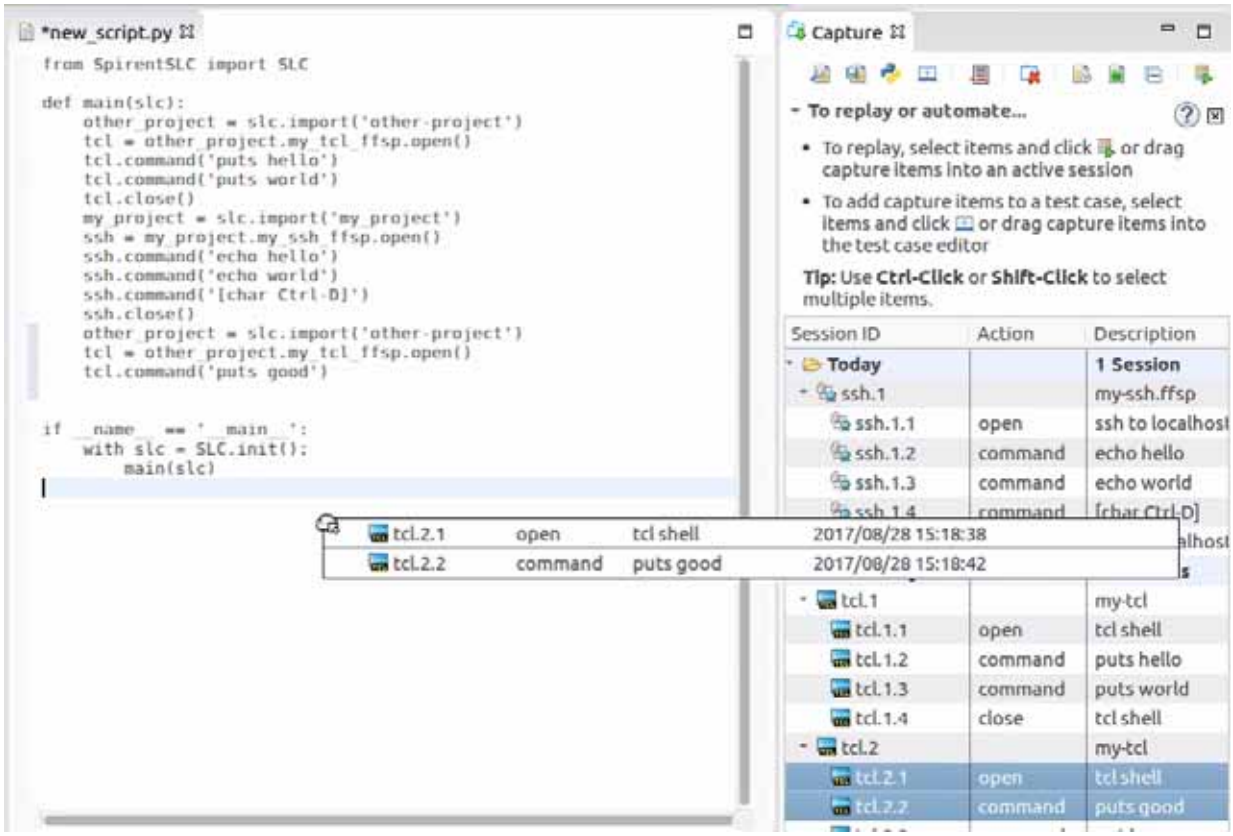
  This is to ensure the commands, for example, REST POST, that include custom step properties to be complete, are also included in the Python Script.

Note The content of the script conforms to a logical mapping from the XML content to their corresponding commands in Python syntax.

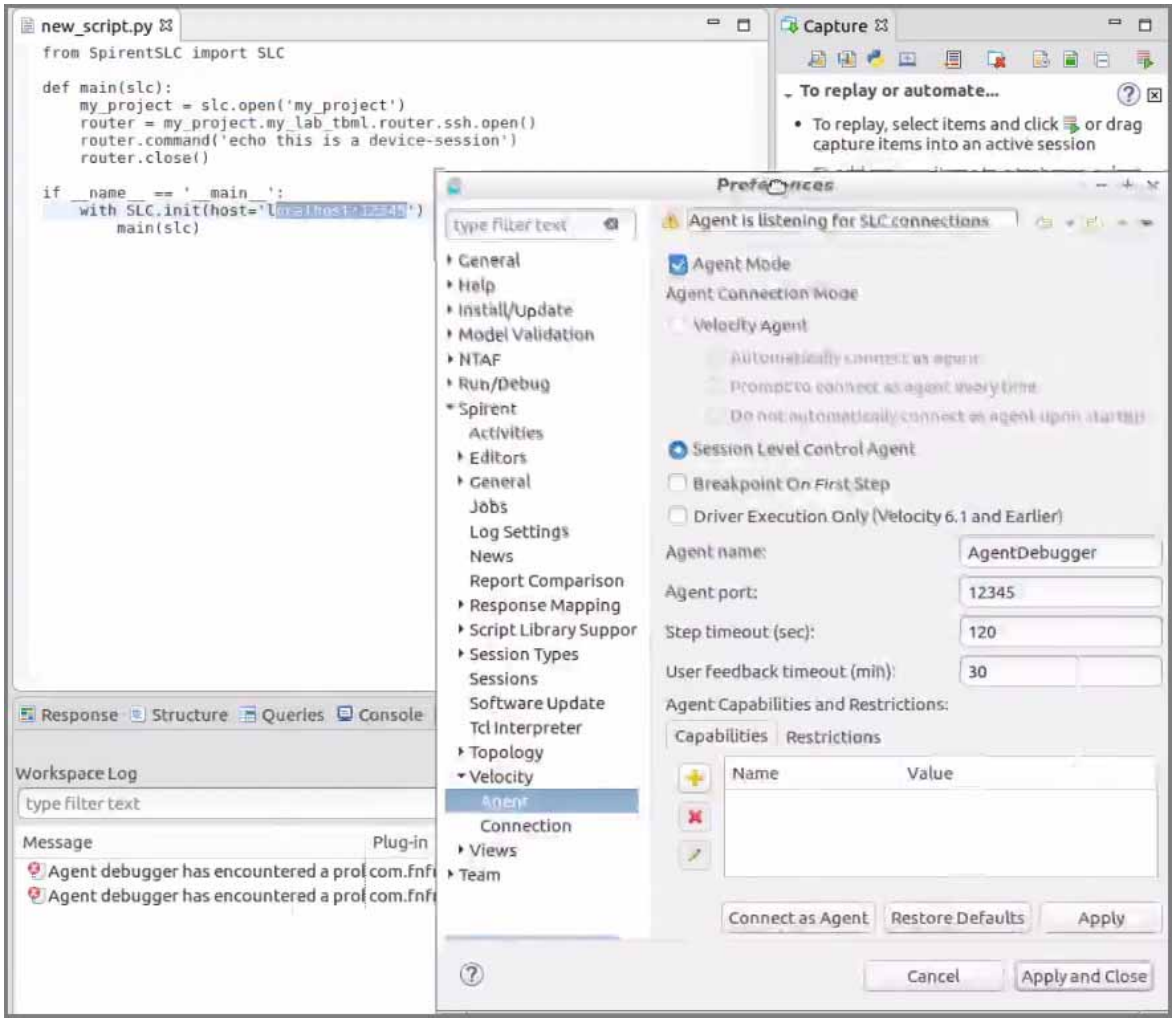
- The selected captured steps are converted into Procedures with the name specified by you in the **Add to Python Script** Wizard.

- iTest converts regular session actions and quick-calls (e.g., print_hello quick-call for tcl-session).
- The Python script generator supports sessions with individual profiles and sessions associated with topology devices (e.g., router session initialization).
- Script will also open all required projects.
- Variables of different sessions with the same profile will have unique names.

You may drag step(s) into an open file in the text editor. In this case, only the steps will be inserted into the Python Script file and not the import command and the **init()** method.


*new_script.py
from SpirentSLC import SLC
def main(slc):
other_project = slc.import('other-project')
tcl = other_project.my_tcl_ffsp.open()
tcl.command('puts hello')
tcl.command('puts world')
tcl.close()
my_project = slc.import('my_project')
ssh = my_project.my_ssh_ffsp.open()
ssh.command('echo hello')
ssh.command('echo world')
ssh.command('[char Ctrl-D]')
ssh.close()
other_project = slc.import('other-project')
tcl = other_project.my_tcl_ffsp.open()
tcl.command('puts good')
if __name__ == '__main__':
with slc = SLC.init():
main(slc)
Capture
To replay or automate...
To replay, select items and click or drag capture items into an active session
To add capture items to a test case, select items and click or drag capture items into the test case editor
Tip: Use Ctrl-Click or Shift-Click to select multiple items.
Session ID
Action
Description
Today
1 Session
ssh.1
my-ssh.ffsp
ssh.1.1
open
ssh to localhost
ssh.1.2
command
echo hello
ssh.1.3
command
echo world
tcl.2.1
open
tcl shell
2017/08/28 15:18:38
tcl.2.2
command
puts good
2017/08/28 15:18:42
tcl.1
my-tcl
tcl.1.1
open
tcl shell
tcl.1.2
command
puts hello
tcl.1.3
command
puts world
tcl.1.4
close
tcl shell
tcl.2
my-tcl

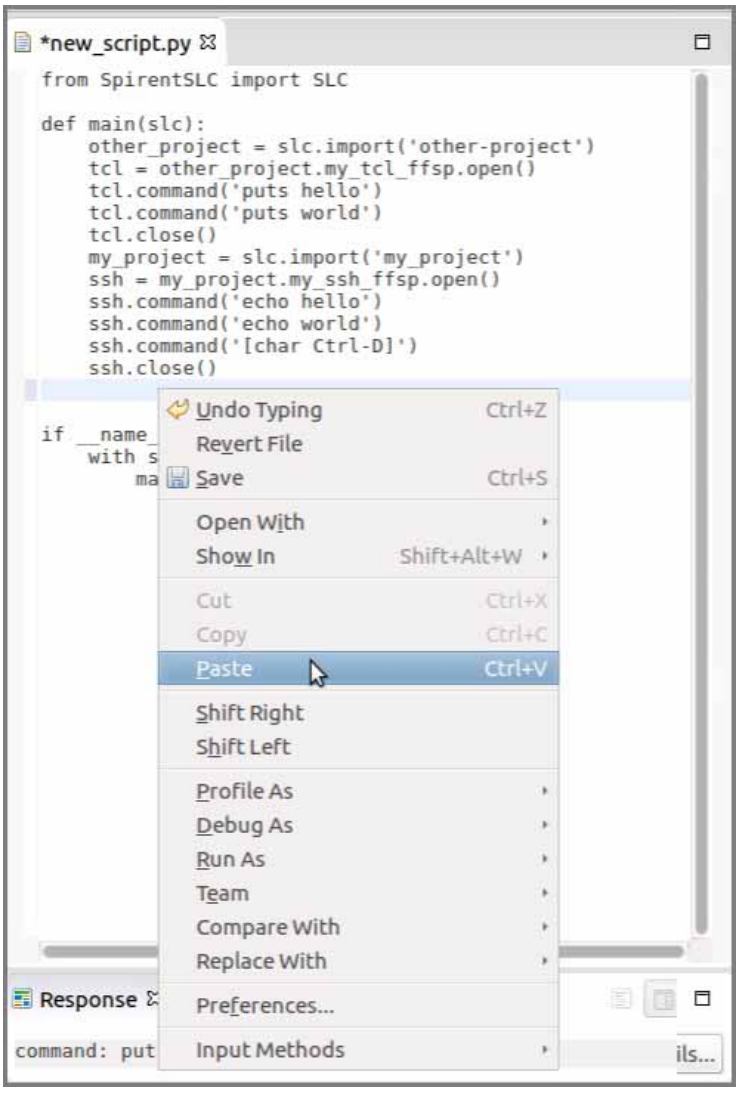
By default, SLC initialization code connects to the iTest GUI on localhost, for example **SLC.init(host="localhost:9005"**). You may remove **host="localhost:9005"** in your Python script editor if you wish to connect to the background agent. The port ID number comes from the Preferences settings.



Copy as Python

Clicking this option generates a number comes from the Script only for the steps and does not include any additional initialization code.

iTest will render the selected steps (open steps, quickcall steps, and native command steps) in Python syntax and copy the line(s) to the clipboard. The contents of the clipboard includes only the script lines associated with the selected steps and will not include the import line and the **init()** line. You may insert these as appropriate into your Python script.



Edit Python Script

You may edit the generated Python Script and include the Python Automation Library commands as required. See “Basic Python Automation Library Commands” on page 1177.

CHAPTER 56

SSH Sessions

SSH session window

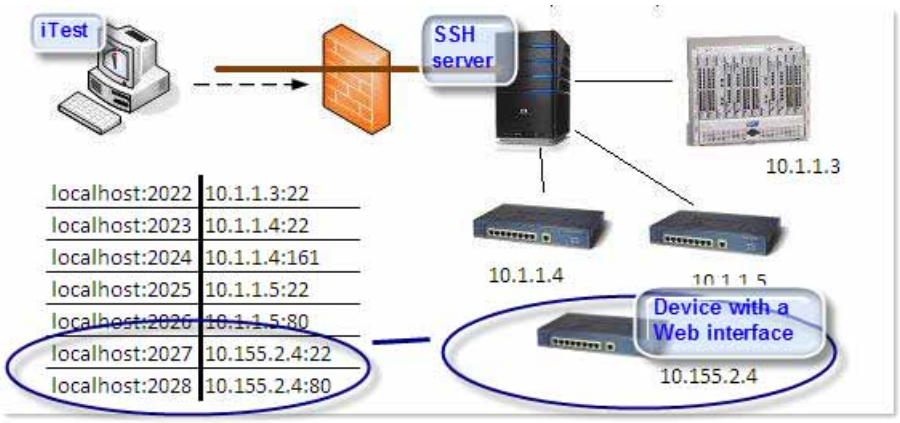
The SSH session window displays your commands and the device's responses. You can think of the editor as a terminal — a terminal that iTest is monitoring and capturing.

Using SSH Local Port Forwarding to connect iTest from your desktop through a firewall to your lab devices

Using iTest over a VNC connection to try to connect to lab devices through the lab firewall is difficult (wrong screen dimensions, latency, no tooltips, and so on) and can require complex setup and troubleshooting. Instead, use iTest's SSH **Local Port Forwarding** capability for a very clean connection.

Example

Here is a typical situation: You want to use iTest from your desktop, but the lab is firewalled. In this example, we will set up a iTest session with a device in the lab (10.155.2.4) that uses both a Telnet interface and a Web interface. The table lists the ports that we have decided to use and will configure on the session profiles for both the SSH server and the Web device.



Step 1 Configure a session profile for the SSH server

1 Create a session profile for the SSH server. On the **SSH** properties page, specify the IP address of the SSH server (iTest can automatically assign an available port on the local host).

2 On the **More > Local port forwarding** properties page, configure the SSH session with a port forwarding list:

Check the box to **Enable local port forwarding**.

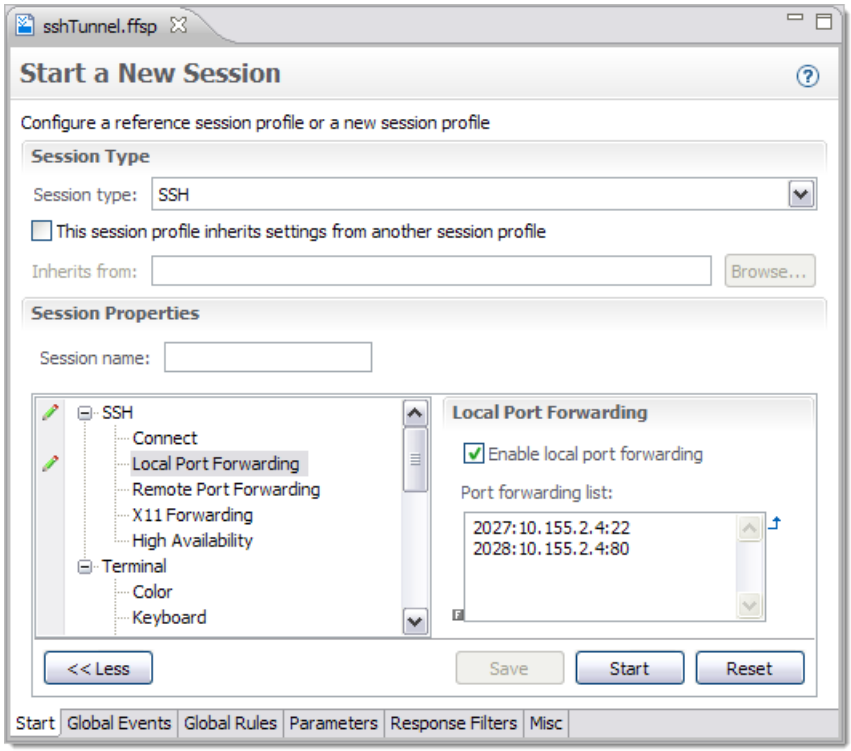
Define a port for each device beyond the firewall that you will connect to. For each port forwarding pair, provide host and port information in the following format (one pair per line):

[localIPaddress:]localPort:remoteIPaddress_or_hostName:remotePort

Notice that **localIPaddress:** is optional.

Field substitutions are supported in any part of the text.

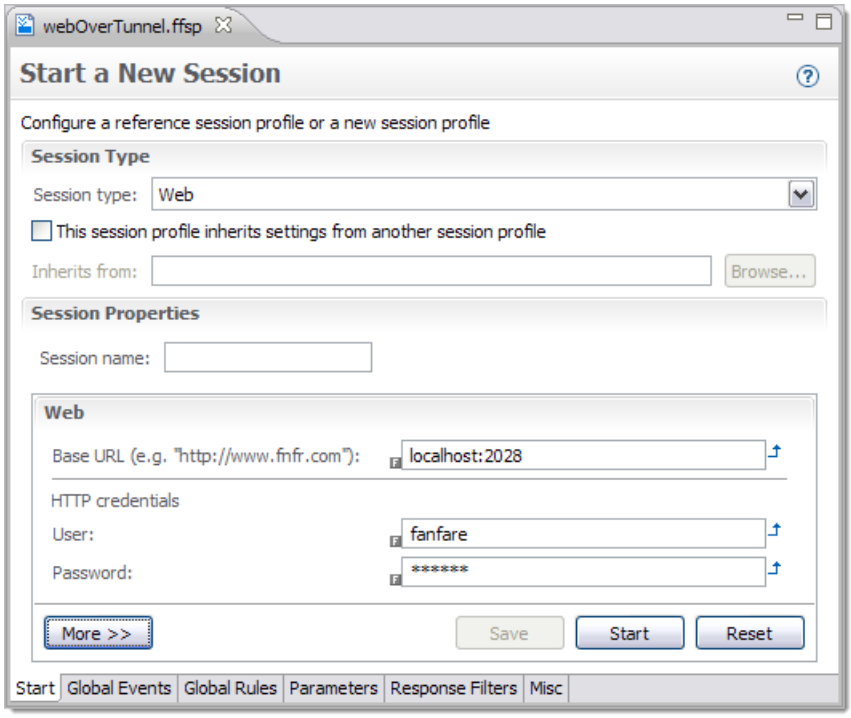
In our example, for the router at 10.155.2.4, we specify that traffic on port 2028 (from the SSH server that is communicating with iTest) should be forwarded to port 80 (typically HTTP traffic) on the router. We also specify that traffic on port 2027 should be forwarded to port 22 (Telnet)



Step 2 Configure a session profile for each device behind the firewall

1 In our example, we create a session profile for the router with the Web interface.

2 We specify the **Base URL** as the device's IP address and the port number that it uses for Web traffic with the SSH server.



Tips for interactive sessions

Note Interactive commands are not supported. For example, Start-Service, Stop-Service, Restart-Service, etc., actions that read from console are not supported.

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.

- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

Session profile property settings for SSH sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

SSH

IP address	Required. Specify the IP address or hostname for the session with the remote host.
Port	Required. Specify the port for the session (number between 1 and 65535). Default: 22
User and Password	Required. Specify the username and password used to connect to the remote host.
SSH version	Required. Specify the SSH version. You must further specify authentication settings on the **SSH authentication** property pages. Default: Auto Auto: When iTest connects to the SSH server, they negotiate to determine the SSH version that they both support SSHv1: SSH Version 1 (not recommended) SSHv2: SSH Version 2

Advanced Authentication

Click **Advanced Authentication** to view and configure SSH Authentication, password,

SSH authentication	Required. Specify the type of authentication that the server allows. When you specify an authentication type, you must specify further authentication settings, as described in the following tables. **Password** (SSHv1 or SSHv2) (default) **KeyboardInteractive** (SSHv2) **PublicKey** (SSHv2) **HostBased** (SSHv1 or SSHv2) **ChallengeResponse** (SSHv1)

SSH authentication: Password

Password	Specify the password used to connect to the remote host. By default, the text is encrypted (masked) here and in all locations where it is used.
Use credentials file	If you configure the **Password** authentication type, then you have the option to configure iTest to go to a specified text file to obtain the latest correct credentials. **Note** The authentication file is not encrypted. Check the box to use the credentials that appear in the file that you specify in the **Credentials file** property. When the box is checked, the settings of the **User** and **Password** properties are ignored.
Credentials file	Specify the path and filename of the text file that holds the credentials to use to log in. **<username>**[space]<**password>** **Note** In the text file, if there is any blank or extra space before the credentials, iTest displays an error after starting the session. See the **Use credentials file** property.

SSH authentication: Keyboard Interactive

Answers	Specify the answers to use when responding to the server challenge. To specify multiple answers, click **Details** and type one answer per line. The text is masked here and in all locations in which it is used.

SSH authentication: PublicKey

Key file	Specify the path to the client key file. Specify a URI that starts with **file://** or **project://**
Passphrase	Optional. Specify the passphrase to use to access the private key in the key file. The text is masked here and in all locations in which it is used.

SSH authentication: HostBased

Key File	Specify the path to the client key file. Specify a URI that starts with **file://** or **project://**
Passphrase	Optional. Specify the passphrase to use to access the private key in the key file. The text is masked here and in all locations in which it is used.

Client hostname	Required for SSHv2 Optional for SSHv1 Specify the hostname of the client
Client username	Specify the user name on the client's computer. For example, you have logged in to the client as username **MyName** and want to connect to the remote host with username **Spirent**. Specify **Spirent** as the **User** property value and **MyName** for **Client username**.

SSH authentication: ChallengeResponse

Challenge answer	Specify the answer to provide when responding to the server challenge. The text is masked here and in all locations in which it is used.

Prompt

For an overview on how prompts work, see "Overview: Prompts in iTest" on page 463.

For instructions on using the properties in this group to define prompts, see "Editing prompt definitions" on page 467.

For related prompt properties, see "Terminal > Replay > Step Defaults > Completion" on page 1131.

Name	The name helps you to remember the type of prompt, for example, **LoginPrompt**. Default: [various]
Content	Specify the exact text of the prompt. **Note**: All prompt definitions are case-insensitive and leading and trailing whitespace is trimmed from any prompt text before iTest attempts to determine whether response text is a prompt. If you use regular expressions in the **Content** value, then set the **Type** property to **Regex**. If the prompt includes a space character or any whitespace in the body of the text, be sure to set the **Type** property to **Wildcard**. Default: [none]

Type	Specify the kind of prompt. **Normal**: Interpret the text in the Content field as the case-insensitive text that you expect for the prompt. **Wildcard**: Disregard any characters that appear in the location of the * character in the text specified for the **Content** property. The most common application for the **Wildcard** setting is to allow for leading or trailing numeric or UserID characters in the prompt (for example Device02>, Device03>, and so on). If you set **Type=Wildcard**, then only the * wildcard character is allowed within the **Content** string (and no other wildcard characters like?). To use other wildcard characters in the **Content** string, you must use **Type=Regex**. **Regex**: Interpret the text specified for the **Content** property as a regular expression. Default: [none]
Is more prompt **More next command** **More quit command**	The **-- more --** prompt is a common method for allowing command line users to view one screen (page) at a time. Many devices use the space character as the command to move to the next page (and often, the letter **q** to exit the display of the response). To enable your automated test cases to page through data that is displayed one page at a time, iTest can automatically "press the space bar" as often as is required to get to the end of the response. As a result, the device's response to the command becomes a single uninterrupted flow of text that does not include the More text. If the prompt is a page-control prompt (for example - - more - -, then: Select the **Is More prompt** checkbox. In the **More next command** text box, specify the command characters (typically a space character) that cause the next page to appear. By default, a space character appears in the box. In the **More quit command** text box, specify the command that exits the More display and returns to the command line prompt. By default, a **q** character appears in the box. Specify a value for **Terminal** > **Replay** > **Step Defaults** > **More**.

Session Properties > More

SSH > Connect

Connect timeout	Specify how long to wait (in seconds) for the session to start. Default: 30 seconds
Retry count	Specify how often to retry the connection when the connection attempt times out. Default: 1
Seconds between keepalives	Some devices are configured to close a session if no traffic occurs for a specified period. To ensure that the session is not auto-closed, iTest can send keepalive signals during periods of silence on the line. Specify the number of seconds that should elapse between keepalives. Default: 0 (do not send keepalives)

SSH > Local Port Forwarding

Enable local port forwarding	Check the box to enable local port forwarding. If you enable local port forwarding, then you must specify port forwarding pair information in the **Port forwarding list** text box. Default: unchecked
Port forwarding list	If you enable local port forwarding, then for each port forwarding pair, provide host and port information in the following format (one pair per line): **[localIPaddress:]localPort:remoteIPaddress_or_hostName:remotePort** Notice that **localIPaddress:** is optional. Field substitutions are supported in any part of the text.

SSH > Remote Port Forwarding

Enable remote port forwarding	Check the box to enable remote port forwarding. If you enable remote port forwarding, then you must specify port forwarding pair information in the **Port forwarding list** text box.
Port forwarding list	If you enable remote port forwarding, then for each port forwarding pair, provide host and port information in the following format (one pair per line): **[remoteIPaddress:]remotePort:localIPaddress_or_hostName:localPort** Notice that **remoteIPaddress:** is optional. Field substitutions are supported in any part of the text. Default: unchecked

SSH > X11 Forwarding

Enable X forwarding	Check the box to enable X forwarding. If you enable X forwarding, then you must identify the display in the **X display location** text box. Default: unchecked
X display location	If you enable X forwarding, then type the information that identifies the display in the following format: **[hostName]:displayNumber** Notice that **hostName** is optional. Field substitutions are supported in any part of the text. If you do not specify a value, then iTest uses **127.0.0.1:0** Default: [blank]

SSH > High Availability

For details on implementing tests for HA devices, see Chapter 38, “Testing High-Availability (HA) Devices”.

High Availability	Check the box to enable HA operation. (The default setting, unchecked, specifies normal, non-HA operation.)
Additional connections	Specify the IP address and port pair for each redundant node (nodes other than the master node.). This information is used only by the **open** step for a session. The values in the list represent nodes 1, 2, 3, ... n. Use the following format, one node per line: **<IP_or_hostname>:<portnumber>** **Important**: Be sure not to enter the values for node 0 — the master node — those values are specified by the **IP Address** and **Port** properties.

Step Defaults > Command

Send interval between each character	Some devices require a delay between characters in order to receive commands correctly. If required, specify the interval of time to wait before sending each character in a command. Default: 0 milliseconds

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Keyboard

Backspace	Specify the code that the device interprets as the backspace character. Default: Ctrl-H
Enter	Specify the code that the device interprets as the Enter character. Default: \r

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Style

Note For session profiles that will support TL1 devices, see "Configuring sessions and test case steps for TL1 devices" on page 1215.

Style	Specify the expected format of the responses to commands. **Normal**: The responses will be text (either structured or not structured). **TL1**: The responses will use TL1 formatting. Default: Normal

Terminal > Replay > Step Defaults > Command

Send interval between each character	Some devices require a delay between characters in order to receive commands correctly. If required, specify the interval of time to wait before sending each character in a command. Default: 0 milliseconds

Terminal > Replay > Step Defaults > Terminator

Line terminator	The settings enable you to configure non-standard terminator settings for a session's behavior during replay. For example, special commands like **show?** do not require the user to press Enter on some devices, but other devices do require the user to press Enter. Typically, you do not need to make any changes to this setting. Default: The default setting (**\r**) is carriage return.
Custom line terminator	Specify the character that indicates the end of a line. Default: [blank]

Terminal > Replay > Step Defaults > Response

Treat LF as CRLF	Check the box to cause iTest to change “LF” (linefeed) characters that are returned by the session into “CRLF” (carriage return / linefeed). This is helpful for sessions that send line endings as “LF” only (for example, Python shell). Default: unchecked
File to write response to	Specify the URI of a file to write the responses into. You can use field replacements in the text of the URI to allow the test case to set the filename at runtime. For example, `file:subdirectory_name/[param file_to_create]` Default: <none> **Note** The full text of the response is written to the file, regardless of the **Number of lines to keep** setting. See the following associated properties: **Append response to file** **Response header** **Number of lines to keep** **Write echo to file** **Write prompt to file**
Response header	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: You may want to specify a text string that should appear before each block of response text. For example: +-+-+-+-+- Next Response Starts Here -+-+-+-+-+ Default: <none>
Number of lines to keep	For very long responses, you might not want to keep all of the response text (as displayed in the Response view for the selected step while working in the Test Report editor or in the Test Case editor). Specify the maximum number of lines to keep for any single response. Specify **0** (zero) to keep all lines. **Note** If you specify a URI in the **Filename to write response to** property, then all lines in the response are written to the file, regardless of this setting. Default: 10,000

Append response to file	If you save responses to a file by specifying a value for the Filename to write response to property, then: Check the box to append each new response to the file specified in the **Filename to write response to** property. Uncheck the box to replace the text of the file specified in the **Filename to write response to** property with the most recent response. As a result, the file will hold only the last response in the session. See the **Filename to write response to** and **Response header** properties. Default: checked
Write echo to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: Check the box to include any echoed characters in the saved response. Default: checked
Write prompt to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: • Check the box to include the last line of the response in the saved response (in command-line applications, this is typically the prompt after the response). • Uncheck the box to not save the last line of the response to the file (the prompt at the beginning of the response where the command was typed is still saved). Default: checked

Note

Options **Write echo to file** and **Write prompt to file** do not work in capture mode. These options are available for **Replay** mode only.

The example below shows how the commands/responses are echoed in these scenarios.

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options *are selected*:

```
prompt>command
response text
etc
etc
prompt>
```

Write echo to file

- When **Write echo to file** is *not selected*

```
response text
etc
etc
prompt>
```

- When **Write prompt to file** is *not selected*

```
prompt>command
response text
etc
etc
```

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options are *not selected*

```
response text
etc
etc
```

This is because the **Terminal > Replay** options are used to replay the captured steps. That is, replay the steps captured via test case execution or replayed from the Capture View ("Working in the Capture view" on page 103).

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal > Replay > Step Defaults > More

Pages to fetch	For responses that are longer than be displayed on a single screen, devices often provide a page-control prompt that enables you to view one screen of text at a time (for example - - more - -). Specify the number of pages to fetch when the more prompt appears (zero means get all pages). If the setting is non-zero, then iTest retrieves that number of pages and then terminates the output from the session's response by sending the command specified for the More: Quit Command property. Default: 100
Device does not remove more prompt. Remove more prompt from response.	Some devices do not remove the text of the more prompt from the text of the response. (For these devices, you will see the more prompt remain at the bottom of the page even after you press the continuation character — typically the spacebar.) Check the box to eliminate the more prompt text from the response that is saved by iTest. Default: unchecked
Use BELL character to detect end of more pages	Some devices do not remove the more prompt from the screen even after you press the continuation or quit character. Such devices often use the audible bell to alert the user that they have reached the end of the response. Check the box to cause iTest to use the bell as its indicator that the response is complete. Default: checked

Terminal > Replay > Step Defaults > Completion

You use **Completion** settings to define when the execution of a step should be considered complete. The determination of when a step is complete is protocol-specific. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- For some steps, you might have defined analysis logic to examine the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

For CLI protocols, you can specify any of several conditions to define when the step is complete, for example, the existence of certain text in the response or the time elapsed after sending the command. The default setting of the **Completion criteria** property is that the step is complete when:

a The session channel is idle for the time specified by the **Idle channel interval** property

and

b The last line of the response matches one of the prompt definitions specified for the session profile or device.

Idle channel interval	This setting helps in cases where you do not know what response to expect and can use a specified idle time (for example, 100 milliseconds) or when you expect no response whatsoever, for example, when talking to a terminal server. Default: 100
Wait for first character before starting idle	Some devices do not respond to a typed command immediately. This setting enables you to ignore the idle time after the last character of the command is echoed and the first character of the actual response is returned. This way, the delay is not misinterpreted as idle channel time for the purpose of determining completion. Default: True
Completion criteria	**Prompt matches AND device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property and last line of the response matches one of the prompt definitions specified for the session profile. **Prompt matches OR device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property **or** the last line of the response matches one of the prompt definitions specified for the session profile. The following processing order occurs: The step is completed once one of the defined prompts is received. If none of the defined prompts is received or no prompt is defined, then the system waits for the specified **Idle channel interval** time (during which the device sends no response data) and then completes the step. **Device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property. **Completion time has expired:** The step is complete when the time specified by the **Completion time** property has elapsed. If you specify **Completion time has expired**, then the **Idle channel interval** property setting is ignored. **TL1 End of Message**: For session profiles that will support TL1 devices, see "Configuring sessions and test case steps for TL1 devices" on page 1215 Default:**Prompt matches AND device has not sent data during the Idle channel interval**
Completion time	Specify the time interval that must elapse for the step to be complete. To apply this setting during execution, the **Completion citeria** property must be set to **Completion time has expired**.

Where to find prompt	Specify where the prompt in a response normally appears. **Last line** **Last non-empty line** The **Any line** setting is a special case that you can use to detect a change in state for an ongoing response. For example, you can detect a port's connection status based on whether the first character of a ping response is **u** or **s**. Be sure to use wildcard characters as needed in the **Content** property. Default: Last line
Command to send when a step is cancelled	Specify the characters to send when the user cancels step execution. Default: \03 (Ctrl-C)
Capture only the last screen of response text	Use this property when you expect a very large response, but the only data of interest appears at the end of the response text. Check the box to cause iTest to save only the last screen of response text. Default: Unchecked
Unknown Prompts During Automated Execution For an overview on how iTest recognizes prompts, see "Overview: Prompts in iTest" on page 463. For instructions on defining prompts, see "Editing prompt definitions" on page 467.	
Expected maximum Idle channel interval	The time to wait for a prompt during automated execution. When this time is reached, iTest displays a **Learn this prompt** link in the status bar. • If the user clicks the link, then iTest opens the **Learn Prompt** dialog box to enable you to add the prompt definition. • If the user chooses not to click the link and the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues. Default: 5
Extra wait before alerting user	Additional time to wait for a prompt during automated execution after the **Expected maximum Idle channel interval** time has been exceeded. When (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed, then: • A countdown timer in the status bar starts to count down the **Time for user to respond** time period. • iTest displays a **Keep waiting** link in the status bar. • If the user clicks the **Keep waiting** link, then iTest waits for an additional (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed period. • If the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues Default: 15
Time for user to respond	Specify the amount of time in seconds to wait for the user to respond once the status bar displays the **Waiting for prompt** timer. Default: 30

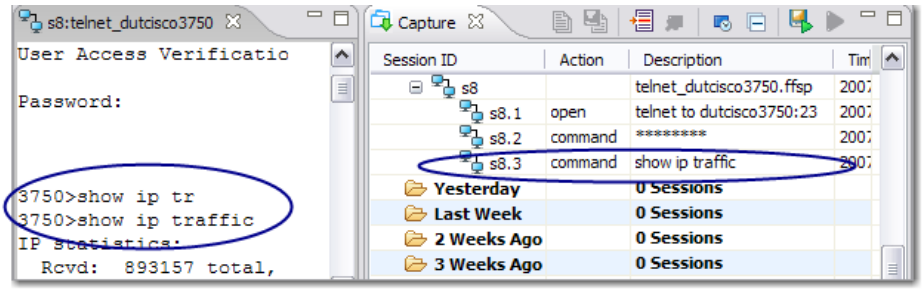
Terminal > Replay > Step Defaults > High Availability

See "Testing HA devices: Detailed instructions" on page 822.

Terminal > Capture

Perform capture cleanup	When you perform manual testing in CLI sessions, you frequently use meta-characters like backspace and up- and down-arrows to correct your typing. In a Capture report, such commands can be difficult to read and understand. If you check **Perform capture cleanup**, then iTest "cleans up" any keyboard shortcuts and removes meta-characters from the captured commands so that the resulting command text appears as if you typed it fully and correctly. See the **Discard command completion steps** property**.** Default: checked A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, even though we actually typed **show ip tr**<tab>. iTest discarded the intermediate **show ip tr**<tab> form of the command.
Mask unechoed commands	Check **Mask unechoed commands** so that, before creating a Capture report, iTest masks all **Command** property text for which no echo was returned. Check the box to automatically mask passwords. Default: checked
Remove echo from response	Check **Remove echo from response** to indicate that the device echoes characters typed at the command line. In this case, iTest ignores echoed characters so that the command text is not added to the actual response text. Default: checked
Remove prompt from response	Check **Remove prompt from response** to save only the response from the session and not the prompt text. We recommend that you do not disable this setting except in rare circumstances. Default: checked
Use prompts from the session for cleanup	Check the box to use the prompt definitions specified in the session profile document when cleaning up commands. Default: checked
Discard command completion steps	To ensure that captured commands in the Capture view are easy to understand, iTest, by default, deletes the intermediate command completion text that was submitted while forming a command. See the **Perform capture cleanup** property. Default: checked
Learn prompts	If the box is checked, then, when you close a session and iTest has detected a new prompt, the Update Session Profile wizard starts. You can specify particular prompts in the **Terminal > Prompts** properties. Default: checked
Learn command completion characters	Learn the character code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can). If the box is checked, then, when you close a session and iTest has detected a new command completion character, the Update Session Profile wizard starts. The default completion character is tab. To specify particular characters, configure the **Terminal > Capture > Command Completion** property. Default: checked

Learn break characters	Learn the character code that the device interprets as a break (so you can manually cancel an executing step). The learned break characters are added to the **Command break characters** property. If the box is checked, then, when you close a session and iTest has detected a new break character, the Update Session Profile wizard starts. Default: checked. The default break character is Ctrl-C. **Note** To specify particular break characters manually, configure the **Command break characters** property (in **Terminal > Capture > Break**).
Remove line containing more prompt	Check the box so that, when capturing responses, iTest deletes the lines that include the more prompt. Default: unchecked



Terminal > Capture > Response

Number of lines to keep	Specify the number of lines in the response to keep in the Capture report. Default: 10,000

Terminal > Capture > Command Completion

Specify the code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can).

Default: tab (\t**)**

To determine the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters.

Limitations of the Record feature:

- For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”.
- Function keys are not recorded.

Command completion requires ENTER key	Most devices respond immediately with command completion when they encounter one of the characters specified for the **Command completion characters** property. Set this property to TRUE, if the device, to perform command completion, requires that you press ENTER after typing one of the characters specified for the **Command completion characters** property. Default: unchecked

Terminal > Capture > Break

Number of lines to keep	Specify the character code that the device interprets as a break (so you can manually cancel an executing step). Default: Ctrl-C To add the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters. **Limitations of the Record feature:** • For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”. • Function keys are not recorded.

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

CHAPTER 82

Swing Sessions (Obsolete and Deprecated)

Capturing Swing sessions

Important

The Swing sessions are ***obsolete and no longer supported*.**

If you open a session, test case, or topology that uses Swing session, the following warning message displays in the Error Log View:

Swing session is deprecated, please use Ranorex or eggPlant session instead.

This chapter is provided only to support existing implementations.

Note To use Swing sessions in iTest on RHEL 5, you must set the PATH environment variable for Java to: `PATH=/usr/java/jre<version>/bin`, for example, `PATH=/usr/java/jre1.6.0._06/bin`

You can use a iTest Swing session to test an application-under-test (AUT) — a Java application with a user interface that was developed using Swing.

Because it is so easy to mistype the paths to the components (controls) on Swing pages, you typically save a captured manual test as a test case rather than manually typing new Swing steps in the Test Case editor.

Actions that are not auto-captured

In nearly every case, you can perform an interactive (manual) test in the Swing application and iTest will capture each step. You then save the captured steps directly into a test case. There are particular steps that iTest cannot capture (listed in “Limitations of Swing auto-capture” on page 1548). For such steps, you will follow the instructions that appear in “Testing Swing steps that cannot be auto-captured” on page 1549.

Launching Swing Applications from iTest

In most cases that a test or interactive session needs to connect to an Java application under test (AUT), the AUT is launched by a batch file. Even if you usually launch the AUT by invoking Java directly from the command line, We recommend that you create a batch file to make it easier to develop test cases.

There are several options for testing Swing applications. See “Swing” on page 1556.

Swing session action reference

The following actions are available in Swing session steps in test cases. To view the set of properties associated with an action, see “Step Properties section: Swing Step Defaults properties” on page 1561. Targets are discussed in “About targets and form maps” on page 1553.

Action	Target	Command	Step Description	Notes Regarding Capture
click	Optional	left/right/middle If blank, then left-click	Performs a mouse click ** The target is optional if the click is performed on absolute coordinates. You can also specify coordinates or keyboard modifiers.	**Limitations** Shift+right-click is reserved for highlighting. To perform the Shift+right-click operation, use the Swing session window, as described in “Swing session action reference” on page 1543. If the AUT does not implement a Ctrl+Shift+right-click context menu, then Ctrl+Shift+right-click opens the iTest context menu, as described in “Testing Swing steps that cannot be auto-captured” on page 1549.
clickCell	Required	Row and column of the cell to click	Clicks the specified cell in a table	iTest does not capture all instances of **clickCell**. iTest captures the following types of **clickCell** actions: • When the cell is already selected and you click it again • When clicking the cell causes a control to be created.
clickColumnHeader	Required	Column in a table	Clicks the specified column header in a table. Typically used to sort a column.	Captured by name. Captures only **clickColumnHeader**, not **click**.
clickItem	Required	Item to click	Clicks the specified item in a list or clicks a tab in a tabbed pane	
clickNode	Required	The path to the node	Clicks the specified node in a tree	Test does not capture all instances of **clickNode**. iTest captures the following types of **clickNode** actions: • When the node is already selected and you click it again • When clicking the node causes a control to be created.
collapseNode	Required	The path to the node ***	Collapses the specified node in a tree	

contextMenu	Required: The item to right-click to bring up the menu	The menu items to select Path to the node by label or by index.	Selects the specified item in a context menu (typically, a right-click context menu). ** If an individual item contains a space, then enclose it in double quotes. For example, to select the **Banana** item in the **New Flavor** menu, use the following **Command**: **"New Flavor"/Banana**	
contextMenuCell	Required	The cell's row and column followed by the path of the context menu item. For example, "Mike" "First Name" "Open With/Text Editor".	Acts on context menus for a cell in a table. Only the context menu for a single cell is supported.	
contextMenuItem	Required	The item's label or index followed by the path of the context menu item. For example, "item 1" "Open With/Text Editor".	Acts on the context menu for an item in a list or a tab in a tabbed pane. Specify the path to the node by label or by index.	Not captured
contextMenuNode	Required	The node's label or index followed by the path of the context menu item. For example, "my_folder/item 1" "Open With/Text Editor".	Acts on context menus for a node in a tree. Specify the path to the node by label or by index.	
describe	Required	N/A	Returns all possible properties of the specified control	
describeCell	Required	Row and column of the cell	Retrieves the properties of the specified cell item in a table	
describeColumnHeader	Required	Column in a table	Retrieves the properties of the specified column header	
describeItem	Required	Item	Retrieves the properties of the specified item in a combo box, list, or a tab	
describeNode	Required	The path to the node ***	Retrieves the properties of the specified node in a tree	
doubleClick	Optional	left/right/middle If blank, then left-click	Performs a mouse double-click. ** The target is optional if the double-click is performed on absolute coordinates.	The modifiers right/middle/left, Ctrl, Alt, and Shift are captured.

doubleClick Cell	Required	Row and column of the cell to click	Double-clicks the specified cell in a table. Typically used to put a cell in edit mode.	Uses the same criteria for specifying the cell to capture as are used for the **selectCells** action.
doubleClick ColumnHeader	Required	Column in a table	Double-clicks the column header of the specified column in a table	Captured if two clicks happen within the operating system's setting for the double-click interval.
doubleClickItem	Required	Item to double-click and button to use (left/right/middle)	Double-clicks the specified item in a list or tab in a tabbed pane	If no button is specified, then left-double-click
doubleClick Node	Required	The path to the node ***	Double-clicks the specified node in a tree	Captured when you double-click the node (not the expansion toggle). This is lower in priority than the other tree actions.
expandNode	Required	The path to the node***	Expands the specified node in a tree	Captured when you expand the node in a tree either by clicking the expansion toggle or by using the keyboard (left/right arrow keys). Captures the node path, not the node indexes.
getItems	Required	N/A	Lists the items in the specified combo box, list, or tabbed pane.	
getTable	Required	N/A	Gets a structured and textual representation of the specified Swing table. Typically used for analysis.	
getText	Required	N/A	Extracts the text of the specified text component and puts it in the response	
getTree	Required	N/A	Retrieves a textual and structured representation of the specified tree	
keyDown	Optional	The key to press	Simulates a key press*	
keyUp	Optional	The key to release	Simulates a key release *	
menu	Required The menu-bar that contains the menus	The menu items to select, separated by forward slashes	Selects a menu item ** If an individual item contains a space, then enclose it in double quotes. For example, to select the **Banana** item in the **New Flavor** menu, use the following **Command**: “New Flavor”/Banana	Captures the final menu item that was selected in a menu bar
mouseDown	Optional	left/right/middle If blank, then left-click	Simulates the ‘button press’ of a mouse button on a control or at the specified coordinates *	Not captured

mouseMove	Optional	N/A	Moves the mouse over a specified control. You can use the **Time to hover** property to specify the number of seconds to hover.	Not captured
mouseOverCell	Required	Row and column of the cell	Moves the mouse over the specified cell	Not captured
mouseOverColumnHeader	Required	Column in a table	Moves the mouse over the specified column header	Not captured
mouseOverItem	Required	Item	Moves the mouse over the specified item in a list or tab in a tabbed pane	Not captured
mouseOverNode	Required	The path to the node ***	Moves the mouse over the specified node	Not captured
mouseScroll	Not required	N/A	Simulates the mouse scroll *	Not captured
mouseUp	Optional	left/right/middle If blank, then left-click	Simulates the 'button release' of a mouse button on a control or at the specified coordinates *	Not captured
scroll	Required	The value to scroll	Scrolls the specified slider or scroll bar. Most often used for sliders, because iTest auto-scrolls scroll bars.	Captures the final location of a slider bar. **Limitation** Actions in scroll bars are not captured.
select	Required	The items to select (space-separated list)	Selects the specified item(s) in a combo box, in a list, or in a tab. If an individual item contains a space, then enclose it in double quotes.	Captures combo box items as strings when the combo item's value has changed. If the combo is editable, however, iTest captures setText instead Captures list selection as a new selection. Captures tab selection, but does not capture tab selection that happens as a result of key mnemonics.
selectCells	Required	Row and column of the cell to select	Selects the specified cell or range of cells in a table	Captured when you select cells using the mouse or keyboard. Captured whether or not the selection was appended to the previous selection. If cells are only deselected, iTest will not capture anything, but if cells are deselected as part of a new selection, then iTest captures the correct new selection. iTest attempts to capture by label name. If there more than 5000 cells or if the column has no name or the name is not unique, iTest captures by index.

selectNode	Required	The path to the node ***	Selects the specified node in a tree. You can also append to the existing selection.	Captured when you select a single node or multiple nodes You can select nodes by clicking them or by using the keyboard when the tree has focus. iTest captures a multi-line command where each line is a node in the resulting selection. For example, if you select one node, the command will be a single line with the path to the node. If you then shift-click and select two more nodes, the command will have three lines. This is analogous to how iTest captures list selection.
selectText	Required	The text to select	Selects the specified text in a text component	
sendKeys	Optional	The string representing keys to send	Simulates typing (keyPress and keyRelease) of any key in a standard 101 style keyboard. The syntax for specifying the key to send is described in: http://msdn.microsoft.com/en-us/library/8c6yea83(VS.85).aspx	
setText	Required	The text value	Sets the specified text in a text component	Captures the text that you typed into a text component. Capture occurs when you click in another location or when you press the Enter key.
snapshot	Not required	N/A	Two actions: Generates an XML representation of the control hierarchy Captures an image of all windows that are open in the application-under-test	

* While in interactive (capture) mode, iTest captures a step, but no action occurs on the AUT (the Swing application-under-test). During replay or test case execution, however, the step executes as expected.

** When the Mouse property is set to **Do not move the mouse when performing this action**, then: While in interactive (capture) mode, iTest captures a step, but no action occurs on the AUT (the Swing application-under-test). During replay or test case execution, however, the step executes as expected.

***To specify the path to a node, use the text value or the index of the node. Start from the root node to the node you want to select and separate the values with a forward slash.

For example, a tree with a root node as **Music** has three child nodes: **Rock**, **Classical**, and **Jazz**. The **Classical** node has a child node called **Beethoven**. Use the following path to specify the **Beethoven** node: **Music/Classical/Beethoven**

Limitations of Swing auto-capture

Note To use Microsoft Internet Explorer 8, you must turn on Compatibility View. (Click the **Tools** button and then click **Compatibility View.**)

Due to the complexity of intercepting Swing actions, there are some Swing steps that iTest cannot capture through direct interaction with the application or that do not replay correctly. In addition, iTest does not capture **X-Y** information during interactive testing.

You can capture such steps using the Swing session window (a specialized tool panel that opens with each Swing session). For instructions on using the Swing session window, see “Testing Swing steps that cannot be auto-captured” on page 1549.

Component / Capability type	Limitation	Workaround
Click	Ctrl right-click is not captured iTest reserves Ctrl+right-click for highlighting the components in the Swing session window Any action involving mouse clicks (click, double click, clickCell, contextMenu, and so on) may not replay correctly on non-Windows platforms if the left and right mouse buttons have been switched Captured clicks are replayed in the middle of the control. If the click has different side effects when it happens in the middle of the control than in the x-y location where it was captured, subsequent steps may fail.	Use the Swing session window to capture
Click in menu	Clicks and modified clicks on menus are not captured	Use the Swing session window to capture
Table	Cell deselection is not captured	
Table	Cell selections made within 500 milliseconds of each other that are appended are not captured	Wait until one cell selection has been captured before selecting another cell.
Table	When cells use a custom renderer and the object in the cell's toString() contains a hash code, replay will fail	Change the class’s toString, or implement ITestable, or change the captured step to click the cell by index.
Tree	Node deselection is not captured	
List	If an entire selection is deselected, iTest does not capture anything. iTest captures a new selection only if part of the selection is deselected.	
Slider	If you scroll the value of two sliders within 500 ms of each other, iTest captures only the second scroll	Wait until one scroll action has been captured before scrolling another slider.
Context menu	iTest does not capture context menus that appear as a result of a key sequence (for example, Shift-F10)	Send keys through the Swing session window to bring up the context menu, then use the **contentMenu** action with the **Do not move the mouse** option selected

Context menu	When you select a menu using key accelerators, but have not made the menu visible, then replay may fail The menu must become visible in order for iTest to capture the correct path.	Rather than using key accelerators, use the mouse to capture menus and context menus.
Disabled controls	iTest does not capture actions on disabled controls	Use the Swing session window to capture
doubleClickItem	doubleClickItem may fail in replay if clicking on the coordinate center of the item does not have the same effect as double clicking on a different part of the item. For example, in the default file chooser dialog, you need to double click an item's label to open it, not the whitespace next to the label	Need to use some other action, such as doubleClick with coordinates specified

Testing Swing steps that cannot be auto-captured

In nearly every case, you can perform an interactive (manual) test in the Swing application and iTest will capture each step. You then save the captured steps directly into a test case and you will not need to follow the instructions that appear in this section.

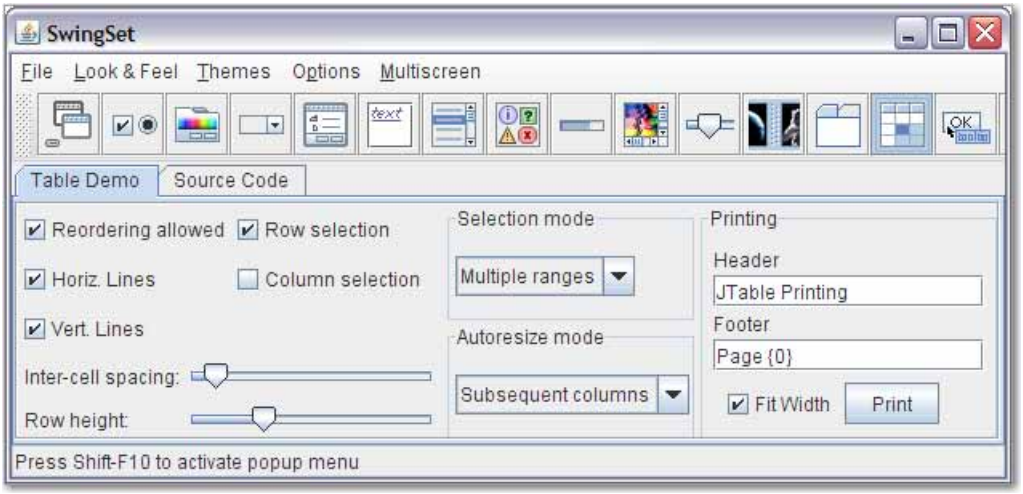
iTest cannot, however, capture some Swing actions (listed in “Limitations of Swing auto-capture” on page 1548). In the following example, we will use the Swing session window (a specialized tool panel that opens with each Swing session) to illustrate how to capture actions that iTest cannot capture directly.

Using the Swing session window to capture particular steps

1 Click Start Capture to prepare to save the manual session as a test case.

2 Start a Swing session from iTest (typically by double-clicking a session profile in the Favorites view).

The Swing application-under-test (AUT) starts. Here is our example application:



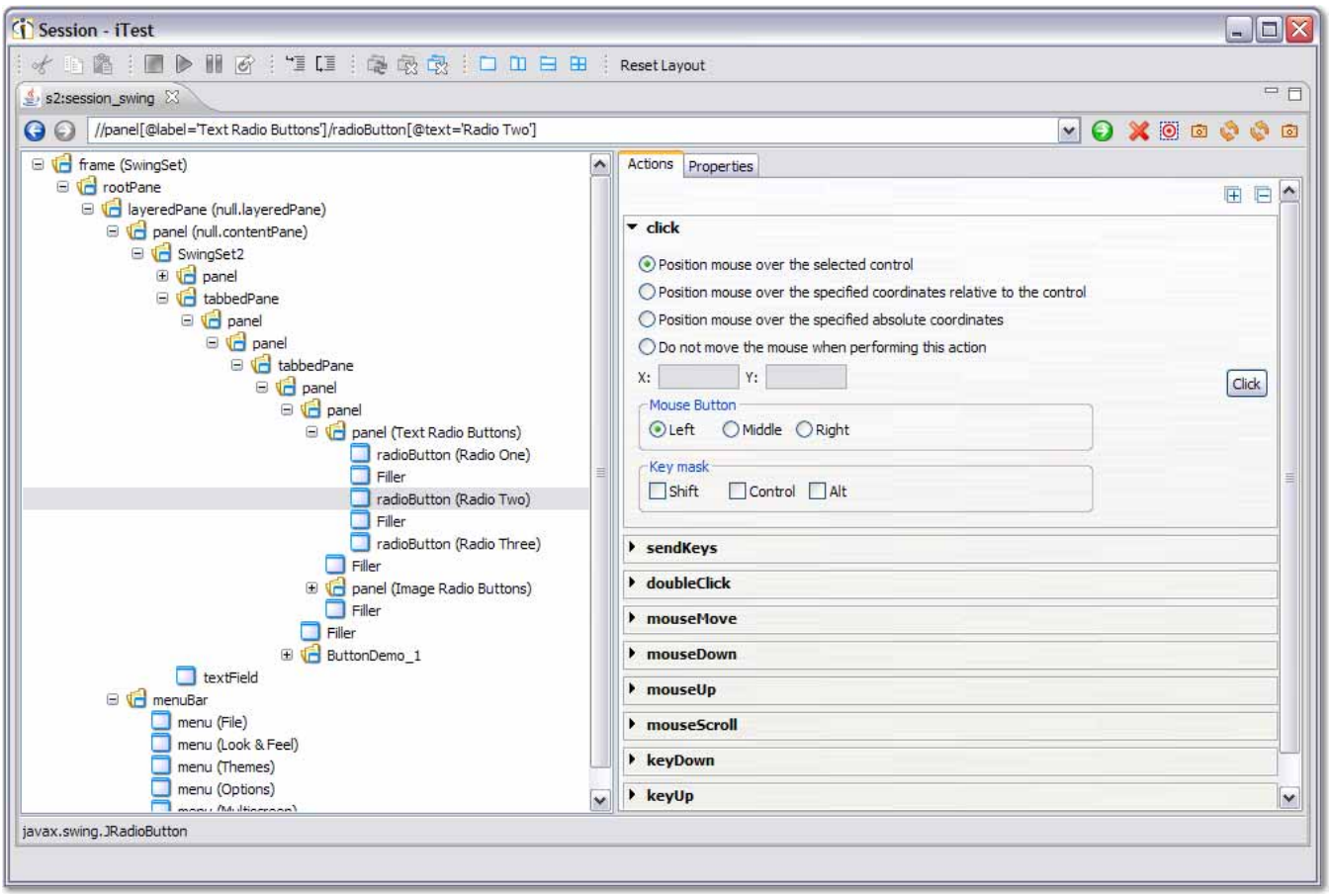
In addition, iTest opens its Swing session window—more about that in a moment.

3 We want iTest to capture the action of typing text into a particular text box.

Swing sessions are captured like any other iTest session—when you click in the text box and type the text, iTest captures your actions. The captured step becomes a step in the resulting test case that will execute exactly as you performed it.

As mentioned earlier, however, iTest cannot capture some Swing actions. For those actions, we instead use the Swing session window to capture all actions (as in this example).

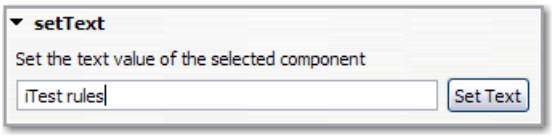
In the AUT, Ctrl+right-click the “Header” text box. iTest highlights the component with a red box for a moment and populates the iTest Swing session window. The Swing session window is split into two panes.



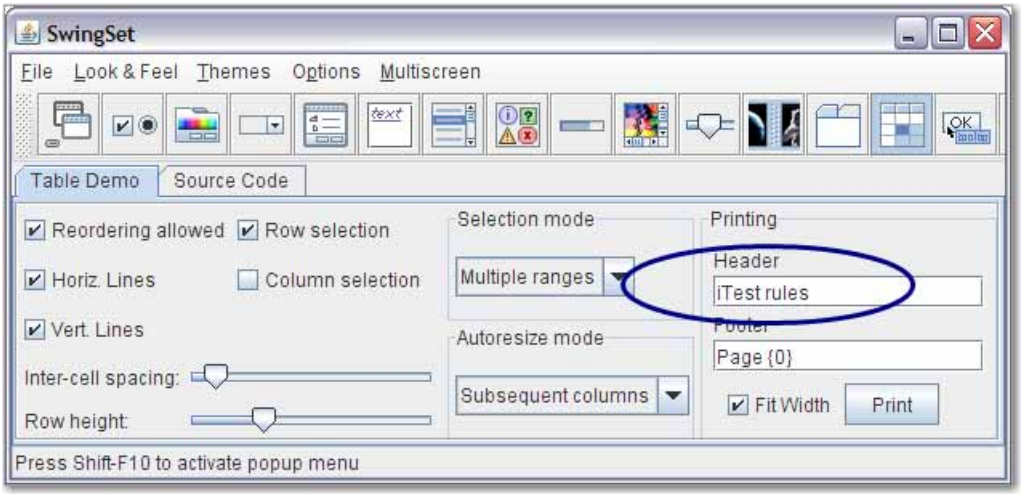
- On the left, you see the ***component tree*** — a structured tree representation of all components on the current page of the AUT. iTest populated the tree when we started the AUT. iTest selected the entry for the “Header” text box component that we Ctrl+right-clicked a moment ago.
- In the address text box at the top, you can see that the XPath to the selected component is **//textField[@label='Header']**
- On the right, in the **Actions** tab, is a list of the kinds of actions that you can perform on the selected component. In the example, we opened the **setText** action group and you can see the kinds of actions you can perform on the text field named **Header**. It looks like there is just one action available: **Set Text**. Some action groups offer several actions. For example, the **doubleClick** group supports right-double-click, left-double-click, Shift+right-double-click, and several other actions.
- If you clicked the **Properties** tab, you would see a structured list of all of the properties of the selected component. More about the **Properties** tab later.
- Notice the status bar at the bottom of the window where you can view detailed information on the currently selected item and progress for ongoing activities.

4 So far, we have captured an **open** session step. To perform an action in the AUT and cause iTest to capture the action, you go to the **Actions** tab, open the appropriate action group, specify the particular action to take, and then click the associated action button.

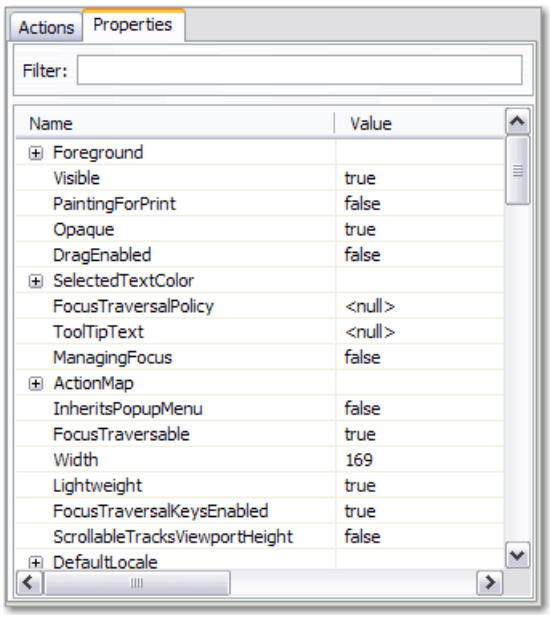
In the example, we open the **setText** action group, type “iTest rules” into the text box, and then click the **Set Text** button. When we click the action button, iTest performs the action (types the text into the field) and captures the step.



Indeed, the text appears in the text box in the application.

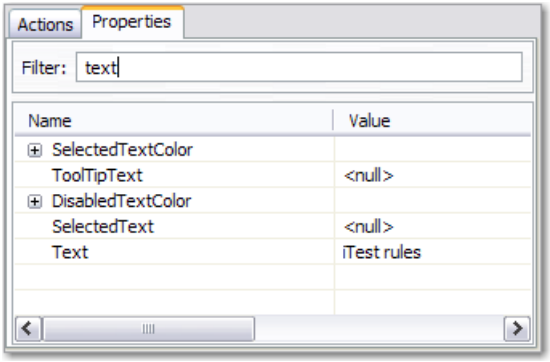


5 In this step, we wish to learn more about the text box in the AUT — let’s capture all of its properties. Click the **Properties** tab in the Swing session window. A structured list of all properties of the text box appears on the page.



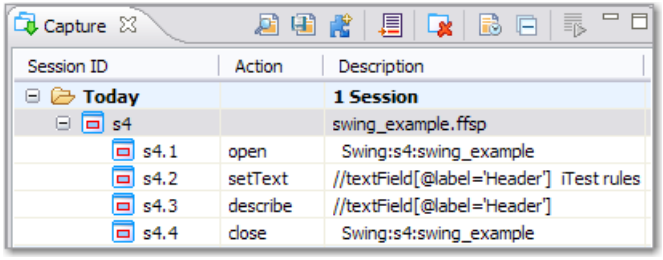
Important Each time you click the **Properties** tab, iTest captures a **describe** action. The response to the **describe** action is the full list of properties and values.

We want to confirm the text value in the box, so let us type “text” into the **Filter** box. Now, only property names and values that include the string “text” appear in the list. The information confirms what we see on the application page: the **Text** value of the text box is “iTest rules”.



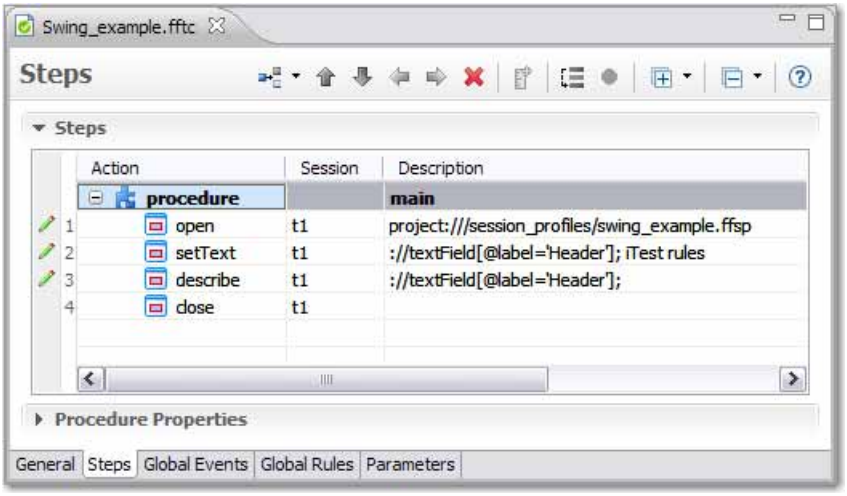
6 Now, we end the session by closing the Swing session window. The AUT window also closes.

7 Let us see what iTest has captured.



Indeed, the **setText** action (that we performed by clicking the action button) and the **describe** step (that we performed by clicking the **Properties** tab) are captured.

8 Click Finish Capture to create the automated test case. The resulting test case looks like this:



Tips Clicking a node in the component tree is equivalent to populating the address bar (based on the tree node) and clicking **Go**. The specified component is selected in the component tree.
Double-clicking a node in the component tree is equivalent to populating the address bar and then clicking **Describe** — the **Properties** tab is loaded.

You can replay captured actions by dropping them from the Capture view into the Swing session window.

About targets and form maps

For Swing-based actions, a target is the entity on the page that is being acted upon. For example, the text box to type into, the button or link to click, or the radio button to select.

While developing a test case, you will make use of the information in a form map to perform an action on a target (for example, to click a particular button on a page).

Targets are described in “Targets” on page 846. Form maps are described in “Overview: Form Maps” on page 843.

Swing session window toolbar

Back	Set the address bar contents back to the previous target. The address bar includes a history of 20 addresses, so you can move through them as needed. Note that a particular target may no longer be valid.
Forward	Set the address bar contents to the next valid target in the session history list following the currently selected target.
Address bar	The current target appears as a valid XPath query. You can type an XPath target string directly, use the drop-down list to select a recent target, use the component tree, or click **Back** and **Forward**.

Go	**Go** evaluates the XPath in the address bar to find a node in the component tree. iTest selects the corresponding node in the component tree and populates the Action page appropriately. Clicking a node in the component tree is equivalent to populating the address bar with an XPath query and then clicking **Go**.
Cancel	**Cancel** the current action immediately.
Describe	The **Describe** button has the effect of selecting the **Properties** tab and causing iTest to capture a **describe** step. The **Properties** page is populated with all properties of the component currently selected in the component tree. Double-clicking a node in the component tree is equivalent to populating the address bar and then clicking **Describe**.
Snapshot	Perform a snapshot of the entire application at this point. Two actions: Return an XML document representing the component hierarchy. Capture a screenshot of the visible windows in the application.
Refresh	Repopulate the component tree according to the application's current state.
Auto-update	You have the option to configure iTest to refresh the component tree periodically as the application's state changes (for example, as new targets come and go). The button toggles between auto-refresh on and off.
Highlight	This option helps you to confirm that you are working on the correct component in the AUT. When you select a component in the component tree, or when you Ctrl+right-click a component in the AUT, iTest outlines the component in the AUT. iTest outlines the component with a red border for two seconds. The outline action is not captured. The button toggles between highlight on and off.

iTest Swing context menu

If the AUT does not have a Shift+right-click context menu, then select an item and click Shift+right-click to open the iTest Swing context menu. Each menu item captures the associated Swing action as described in “Swing session action reference” on page 1543.

Note You can configure the key mask as a session profile property. See “Session profile property settings for Swing sessions” on page 1555.

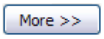
Describe	Captures all properties of the component that was clicked.
Snapshot	Perform a snapshot of the entire application at this point. Two actions: • Return an XML document representing the component hierarchy. • Capture a screenshot of the visible windows in the application.
Get Text	Available if the control is a JLabel or a JTextComponent. Not available for JPasswordField (which is a JTextComponent). Places the text of the specified text component in the response.

Get Table	Available if the control is a table. Returns a structured and textual representation of the table. Typically used for analysis.
Describe Cell	Available if the control is a table. Retrieves the properties of the specified cell item in a table
Describe Column Header	Available if the control is a table. Retrieves the properties of the specified column header
Get Tree	Available if the control is a tree. Retrieves a textual and structured representation of the tree.
Describe Node	Available if the control is a tree. Retrieves the properties of the specified node in a tree
Get Items	Available if the control is a combo box, list, or tabbed pane. Returns the list of the items in the selected combo box, list, or tabbed pane.
Describe Item	Available if the control is a combo box, list, or tabbed pane. Retrieves the properties of the specified item in a combo box, list, or a tab

Session profile property settings for Swing sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Swing

There are several options for testing Swing applications. The method you specify for launching the session determines which properties you specify in the **Swing** property group.

- "'Launch a jar file or Java class' option" on page 1557
- "'Launch a JNLP application' option" on page 1557
- "'Launch a custom application' option" on page 1558
- "'Connect to an existing application' option" on page 1558

In most cases that a test or interactive session needs to connect to an Java application under test (AUT), the AUT is launched by a batch file. Even if you usually launch the AUT by invoking Java directly from the command line, We recommend that you create a batch file to make it easier to develop test cases.

Setting the environment variables

Set the following environment variables using one of the options that follow the table.

JAVA_TOOL_OPTIONS	Set to the arguments that the JVM needs to talk to iTest. The value of the variable should be: -javaagent:{iTest_Swing_resources}/SwingTestHarness.jar -Xbootclasspath/a:{iTest_Swing_resources}/SwingTestHarnessInterfaces.jar;{iTest_Swing_resources}/SwingTestHarness.jar;{iTest_Swing_resources}/xmlwriter-2.2.jar {iTest_Swing_resources} is the path of the Swing resources plugin located in the iTest installation folder. **Note** The path separator for Linux is colon ':' and on Windows is semicolon';' Remember to use quotes as appropriate for your platform.
iTestSwingServerPort	Specify the port to use for iTest communications. You do not need to specify a different port for every application. All applications running on the computer can use the same port. Default: 5030
iTestSwingServerPath	Specify the path to the iTest Swing server launcher. This is located in the {iTest_Swing_resources} folder. {iTest_Swing_resources}/runserver.sh on Linux {iTest_Swing_resources}\runserver.bat on Windows

Option 1: Specify the environment variables in the normal way For iTest to work correctly, you must specify the path for the correct version of Java in environment variables. To verify, type the command **java -version** in a command line console—the system should return the version that you expect.

Option 2: Specify the environment variables in the batch files Alternatively, use the full path to Java in the batch files that you specify in the session properties. If you choose to specify the full path, then you must edit the **swingServer.bat** file (Windows) or **swingServer.sh** file (Linux) to use the full path.

If you do not want to modify batch file for iTest testing, then you can create an alternate wrapper batch file that first sets the environment variables and then calls the original batch file. The environment variables to set are:

‘Launch a jar file or Java class’ option

Select this option if iTest can launch a new session with the Swing application under test using one of the following methods:

- Launching a Jar file
- Launching a Java class.

Location of the jar file to launch	Specify the jar file that will start the application under test. If you specify a value, then do not specify a value for **Class to launch**.
Class to launch	Specify the Java class that will start the application under test. If you specify a value, then do not specify a value for **Location of the jar file to launch**.
Classpath	Optional. Specify the classpath required to launch the application under test.
Working folder	Specify the folder to start the file in.
Additional command line options	Optional. Specify additional options that the application under test might expect when being launched from the command line.

‘Launch a JNLP application’ option

Use the **Launch a JNLP application** option to launch a Java Web Start JNLP application. On the web page, when you click a link to a JNLP file, the Java Swing session starts.

Limitations

If the JNLP application starts a new process (not a thread) then the process is not instrumented and iTest will not be able to capture it. If the JNLP application terminates the initial process (for example the initial process was the launcher for the other application) then the iTest JNLP session will be terminated and you will not be able to connect to the new process.

Workaround Use **javaws** to launch such a JNLP application manually and connect iTest using the **Connect to an existing application** option.

Application URI	Specify the URI of the application under test. file:// project:// and http://schemes are supported
Java Web Start command line options	Optional. Specify additional options (see link) that the application under test might expect when being launched from the command line. http://download.oracle.com/javase/1.5.0/docs/guide/javaws/developersguide/javaws.html

‘Launch a custom application’ option

Use the **Launch a custom application** option if the AUT launches only a single Java process.

Command to launch the application	Specify the batch file. See “Example batch file — launches target Swing application” on page 1560.
Application working folder	Specify the folder to start the batch file in.
Additional command line options	Optional. Specify additional options that the application under test might expect when being launched from the command line.
Auto-instrument the application to launch	To enable iTest to talk to the AUT, the test must pass certain arguments to the JVM used by the AUT. To pass the required arguments, check **Auto-instrument the application to launch**. **Note** In earlier iTest versions, you needed to specify the arguments in the batch file. If you already have such a batch file, then you do not have to do anything else.

‘Connect to an existing application’ option

Use the **Connect to an existing application** option in the following situations:

- You do not want to re-launch the application for every session; that is, you want to start the application and execute various test cases on the same instance of the AUT
- The AUT runs on one computer and iTest runs on another
- The AUT consists of multiple processes or launches child processes

When using this option, you are responsible for launching the application—iTest will connect only to an application that has already been launched. In the session profile, you can specify the hostname of the computer and specify the port to connect to.

In most cases that a test or interactive session needs to connect to an AUT, the AUT is launched by a batch file. Even if you usually launch the AUT by invoking Java directly from the command line, We recommend that you create a batch file to make it easier to develop test cases.

Setting the environment variables Set the following environment variables using one of the options that follow the table.

JAVA_TOOL_OPTIONS	Set to the arguments that the JVM needs to talk to iTest. The value of the variable should be: -javaagent:{iTest_Swing_resources}/SwingTestHarness.jar -Xbootclasspath/a:{iTest_Swing_resources}/SwingTestHarnessInterfaces.jar;{iTest_Swing_resources}/SwingTestHarness.jar;{iTest_Swing_resources}/xmlwriter-2.2.jar {iTest_Swing_resources} is the path of the Swing resources plugin located in the iTest installation folder. **Note** The path separator for Linux is colon ':' and on Windows is semicolon';' Remember to use quotes as appropriate for your platform.
iTestSwingServerPort	Specify the port to use for iTest communications. You do not need to specify a different port for every application. All applications running on the computer can use the same port. Default: 5030
iTestSwingServerPath	Specify the path to the iTest Swing server launcher. This is located in the {iTest_Swing_resources} folder. {iTest_Swing_resources}/runserver.sh on Linux {iTest_Swing_resources}\runserver.bat on Windows

Option 1: Specify the environment variables in the normal way For iTest to work correctly, you must specify the path for the correct version of Java in environment variables. To verify, type the command **java -version** in a command line console—the system should return the version that you expect.

Option 2: Specify the environment variables in the batch files Alternatively, use the full path to Java in the batch files that you specify in the session properties. If you choose to specify the full path, then you must edit the **swingServer.bat** file (Windows) or **swingServer.sh** file (Linux) to use the full path.

If you do not want to modify batch file for iTest testing, then you can create an alternate wrapper batch file that first sets the environment variables and then calls the original batch file. The environment variables to set are:

Tip You can use a iTest Process session or Command Prompt session to launch a process.

Property settings in the session profile Specify the following connection settings:

IP address / hostname	Specify the host that the application is running on.
Port	Specify the port for connecting to the application. This value is the same as the value set for the **iTestSwingServerPort** environment variable. Default: 5030

Application arguments	You do not type in this field. iTest populates the property setting when you select the application after clicking **Browse Active Applications**. See “To connect to an application that is already running” on page 1560.
Browse Active Applications	Click the button to see which applications are currently running on the computer on the specified port. See “To connect to an application that is already running” on page 1560.

To connect to an application that is already running Launch the batch file and then tell iTest to connect to it as follows:

a On the Session Profile editor, click **Browse Active Applications** to see which applications are currently running on the computer on the specified port. Select the application to connect to.

b Start the Swing session to connect to the application. From the connected Swing session, click the **Refresh** button.

Example batch file — launches target Swing application

Use the following batch file to test the **Connect to an existing application** option in iTest:

```
export
JAVA_TOOL_OPTIONS="-javaagent:/home/spirent/Spirent/iTest_4.1/plugins/com.
fnfr.svt.applications.java.swing.resources_4.1.0.54890/SwingTestHarness.ja
r
-Xbootclasspath/a:/home/spirent/Spirent/iTest_4.1/plugins/com.fnfr.svt.app
lications.java.swing.resources_4.1.0.54890/SwingTestHarnessInterfaces.jar:
/home/spirent/Spirent/iTest_4.1/plugins/com.fnfr.svt.applications.java.swi
ng.resources_4.1.0.54890/SwingTestHarness.jar:/home/spirent/Spirent/iTest_
4.1/plugins/com.fnfr.svt.applications.java.swing.resources_4.1.0.54890/xml
writer-2.2.jar"
export iTestSwingServerPort=5030
export
iTestSwingServerPath=/home/spirent/Spirent/iTest_4.1/plugins/com.fnfr.svt.
applications.java.swing.resources_4.1.0.54890/runserver.sh
java -jar SampleSwing.jar
```

Swing > Close Step

Terminate the Swing application when the iTest session ends	Check the box to cause iTest to exit the application under test when the iTest session ends. Uncheck to leave the AUT open. This enables AUTs that perform “cleanup” or other tasks to perform those tasks even when the iTest session end. Default: Checked
Maximum time to wait before ending the session	If you check the **Terminate the Swing application when the iTest session ends** property, then specify the maximum number of seconds to wait after submitting a close step to close the iTest session If the application under test closes at any time, the iTest session will close immediately. Default: 0 seconds

Swing > Target

Targets are described in “Targets” on page 846.

Maximum time to wait for a target	Specify the maximum number of seconds to wait for a target to appear before performing an action on it.
Wait for the target to be enabled	Check the box to cause iTest to wait for targets to be enabled before performing the specified action on the target. Default: Checked
Wait for the target to be visible	Check the box to cause iTest to wait for targets to be visible before performing the specified action on the target. Default: Checked

Robot

Maximum time to wait for the event queue to be idle	**Advanced Users Only** Specify the maximum number of milliseconds to wait for all on-screen side-effects of robot actions to occur (applied before and after all robot actions during test case execution). We recommend that you not change this setting. Default: 2000 (2 seconds)

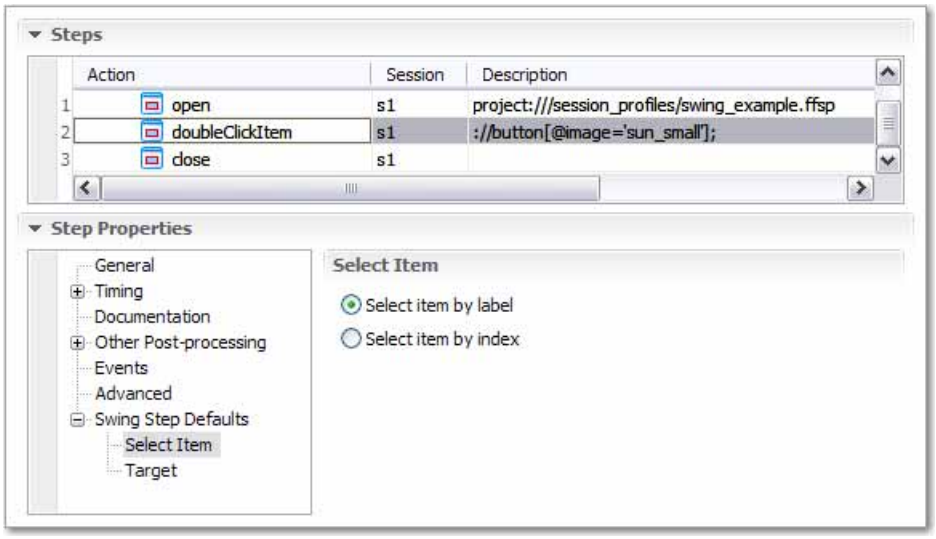
Swing > Capture > Meta-Action

Key Masks	Check all keys that must be depressed while middle-clicking or right-clicking to bring up the iTest context menu during an interactive iTest Swing session. Default: Shift

Step Properties section: Swing Step Defaults properties

Each Swing action has associated groups of property settings. In this example, in the **Swing Step Defaults** group, we view the **Select Item** property group for the **doubleClickItem** action. There are

two settings that you can choose from: **Select item by label** and **Select item by index**. (As with all session types, the **open** and **close** actions are special—properties for these steps enable you to overwrite the property settings that you set in the Swing session profile.)



This table lists the property groups associated with each Swing action. Descriptions for all properties in each group appear after the table.

Action	Property Groups
click	Mouse Properties Target
clickCell	Select Cell Target
clickColumnHeader	Select Column Target
clickItem	Select Item Target
clickNode	Tree Node Target
collapseNode	Tree Node Target
contextMenu	Mouse Button Mouse Properties Target
contextMenuCell	Select Cells Mouse Button Mouse Properties Target
contextMenuItem	Select Item Mouse Button Mouse Properties Target

contextMenuNode	Tree Node Mouse Button Mouse Properties Target
describe	Target
describeCell	Select Cell Target
describeColumnHeader	Select Column Target
describeItem	Select Item Target
describeNode	Tree Node Target
doubleClick	Mouse Properties Target
doubleClickCell	Select Cell Target
doubleClickColumnHeader	Select Column Target
doubleClickItem	Select Item Target
doubleClickNode	Tree Node Target
expandNode	Tree Node Target
getItems	Get Items Target
getTable	Get Table Target
getText	Target
getTree	Get Tree Target
keyDown	Target
keyUp	Target
menu	Target
mouseDown	Mouse Properties Target
mouseMove	Mouse Properties Target
mouseOverCell	Select Cell Target
mouseOverColumnHeader	Select Column Target

mouseOverItem	Select Item Target
mouseOverNode	Tree Node Target
mouseScroll	Target
mouseUp	Mouse Properties Target
scroll	Scroll Properties Target
select	Selection Mode Robot Target
selectCells	Select Cells Target
selectNode	Node Selection Target
selectText	Select Text Target
sendKeys	Target
setText	Target
snapshot	Target

Property descriptions

For all Swing property settings, indexes start with 0 (they are zero-based).

Get Items

Start index	Index of the first item to include in the selection.
Total number of items	Number of items to select (you specify the first item to select using the **Start index** setting).

Get Table

Cell range	This set of properties identifies the section of the table to get values from. Start row / column Number of rows / columns
Key column	This set of properties identifies the candidate key for the table. The key column is used as a key in the queries generated for the table. **Auto**: Automatically determine the key column based on the uniqueness of the values in a column. **Key by column name**: Parse the value of the **Key value** setting as text (the column name). **Key by column index**: Parse the value of the **Key value** setting as the index of the column. **No key**: Do not use a key in the generated queries
Key value	Specify the key for the table.

Get Tree

Maximum depth	Specify the maximum number of levels into the tree to navigate. If the setting exceeds the tree's actual depth, then all levels are selected. Default: 50
Maximum number of rows	Specify the maximum number of rows into the tree to navigate. If the setting exceeds the tree's actual depth, then all rows are selected. Default: 100
Show only nodes that are visible in the current expanded state	Retrieve only nodes that are visible or expanded when the **getTree** step executes. Default: Unchecked

Mouse Button

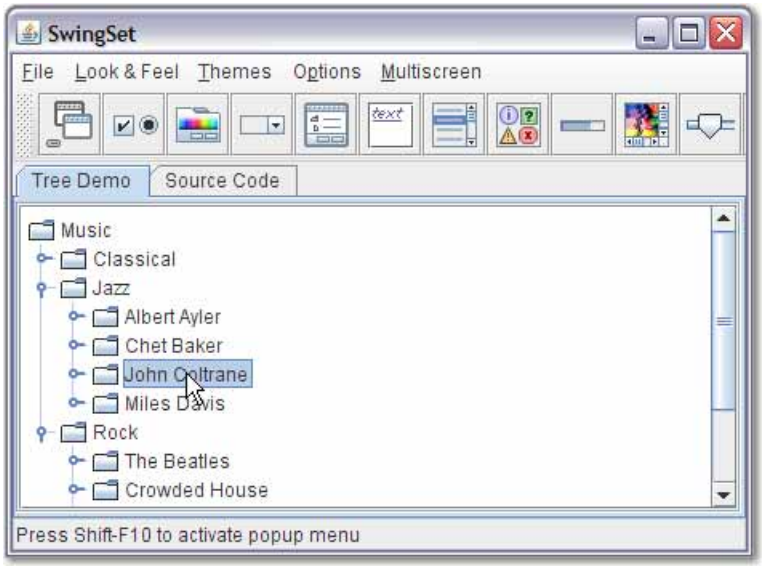
Mouse button to click	Specify the mouse button that opens the context menu. Default: Right button

Mouse Position

Specify how to position the mouse	**Position mouse over the target control**: Place the cursor over the center of the control. **Position mouse over the specified relative coordinates (relative to the top left corner of the control)**: See the **Coordinates** property. **Position mouse over the specified absolute screen coordinate**: See the **Coordinates** property. **Do not move the mouse when performing this action (not applicable for 'mouseMove' action)**: During execution, the action is performed at the current mouse location.
Coordinates	If you specified **relative coordinates over the target** or **absolute screen coordinates**, specify the **X** and **Y** values (pixels). Default: 0,0
Time to hover over a control after 'mouseMove' action	For **mouseMove** steps only, specify the hover time in seconds. Default: 0
Key Masks	Specify keys to press when the click occurs.

Node Selection

Select path by node index	Check the box to identify the node by its index. In the example below this table, the node at 0/1/2 is selected. Uncheck to specify by name (Music/Jazz/John Coltrane) Default: Unchecked
Append to existing selection	Add the specified node to the currently selected nodes. Default: Unchecked



Robot

Simulate mouse and keyboard when executing	Check the box to move the cursor during test case execution. Default: Checked

Scroll Properties

How to scroll	**Scroll by value**: Scroll to the value specified by the **Command** property. **Scroll by percentage**: Scroll to the percent of the entire control specified by the **Command** property. **Increment** / **Decrement:** For sliders, increase (move to the right, typically) or decrease (move to the left, typically) the location of the indicator on the slider by the value or percentage specified by the **Command** property. Page up / Page down: For page control move farther in the document or back in the document. Default: **Scroll by value**

Select Cell

Row	**Row**	This set of properties identifies the row to select. **Auto**: If the specified row identifier is text, then find the first row containing the text. If the specified row identifier is a number, then parse the value as the row index. **By cell value**: Parse the specified value as the contents of the cell. **By index**: Parse the specified value as the row index. **Default**: Auto
	Key Column	Valid only when you specify **By cell value** for the **Row**. This set of properties identifies the candidate key for identifying the row. **Column name**: Parse the value of the **Key** property as the name of the column. **Column index**: Parse the value of the **Key** property as the column index. **Key**: Specify the key value
Column		This set of properties identifies the column to select. **Auto**: Automatically identify the column. If the specified column identifier is text, then parse the value as the name of the column. If the specified column identifier is a number, then parse the value as the column index **By name**: Parse the specified value as text (the column name). **By index**: Parse the specified value as the index of the column. **Default**: Auto

Select Cells

Append to selection	Check the box to add the specified cells to the current selection. Default: Unchecked
Select a cell range	Check the box to specify that the selection is a range of cells and not the default setting of a single cell that is specified by **Start cell**. The **End cell** section is enabled when you check the box. Default: Unchecked

Start cell	Row	This set of properties identifies the row to select. **Auto**: If the specified row identifier is text, then find the first row containing the text. If the specified row identifier is a number, then parse the value as the row index. **By cell value**: Parse the specified value as the contents of the cell. **By index**: Parse the specified value as the row index. **Default**: Auto
	Key Column	Valid only when you specify **By cell value** for the **Row**. This set of properties identifies the candidate key for identifying the row. **Column name**: Parse the value of the **Key** property as the name of the column. **Column index**: Parse the value of the **Key** property as the column index. **Key**: Specify the key value
	Column	This set of properties identifies the column to select. **Auto**: Automatically identify the column. If the specified column identifier is text, then parse the value as the name of the column. If the specified column identifier is a number, then parse the value as the column index **By name**: Parse the specified value as text (the column name). **By index**: Parse the specified value as the index of the column. **Default**: Auto
End cell	If **Select a cell range** is checked, then this section enables you to specify the last cell in the selection range. See **Start cell**	

Select Column

Column	**Auto**: If the specified value is text, then parse the column identifier as the name of the column. If the specified value is a number, then parse the column identifier as the column index. **By name**: Parse the column identifier as the name of the column. **By index**: Parse the column identifier as the column index. **Default**: Auto

Select Item

Select Item	Specify how to select the specified item. **Select item by label**: Parse the specified identifier of the item as its label. **Select item by index**: Parse the specified identifier of the item as its index. Default: Select item by label

Select Text

Select all text	Check the box to select all of the text in the control. Default: Unchecked

Selection Mode

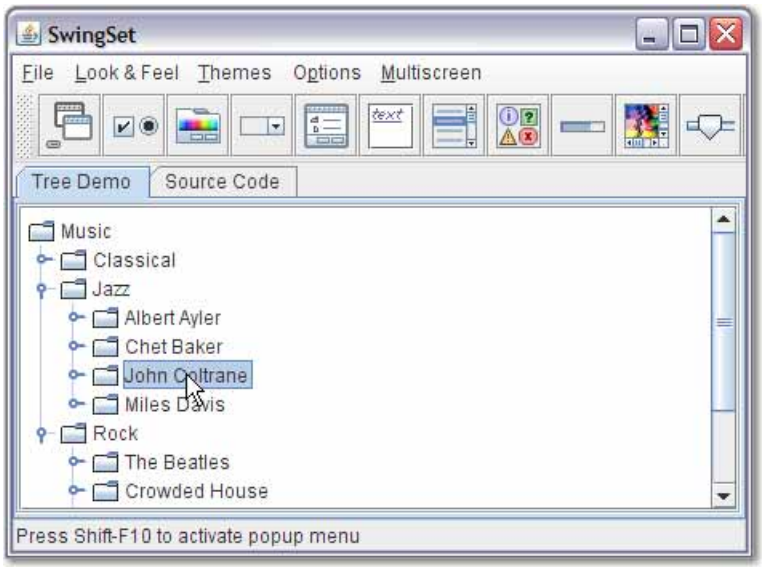
Select item by	Specify how to select the specified item. **Label**: Parse the specified identifier of the item as its label. **Index**: Parse the specified identifier of the item as its index. Default: Label
Append items to existing selection	Add the selected item to the list of selected items.

Target

Maximum time to wait for a target	Specify the maximum number of seconds to wait for a target to appear before performing an action on it.
Wait for the target to be in the enabled state	Check the box to cause iTest to wait for a target to be enabled before performing the specified action on the target. Default: Checked
Wait for the target to be visible	Check the box to cause iTest to wait for a target to be visible before performing the specified action on the target. Default: Checked

Tree Node

Select path by node index	Check the box to identify the node by its index. In the example below this table, the node at **0/1/2** is selected. Uncheck to specify by name (the same example is specified as **Music/Jazz/John Coltrane**). Default: Unchecked



Setting preferences for Swing sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Swing.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Collapse all action groups on the Action tab by default	Check the box to display all action groups on the Action tab with only the title and an expansion arrow. Uncheck the box to display all action groups expanded. Default: checked
Enable debug mode	Useful when debugging. Enables you to monitor messages that are exchanged between the AUT and iTest. Check the box to display all iTest messages in the Console view (To open the view: Click **Window > show View > Other**. In the **General** group, click **Console**.) Default: unchecked

Making Java Swing custom controls available to iTest

In the Swing session window, iTest represents all UI components on the current page of the AUT (its DOM) as a structured tree. The tree is updated dynamically as various controls appear and disappear. When you select a component in the tree, the **Actions** tab displays the list of actions (also called capabilities) that you can perform on the selected component.

iTest generates the list of capabilities in the **Actions** tab based on the type of control. If your application uses a custom control, iTest will try to determine the base component that the custom component derived from and then associate the actions for that control. This strategy works for most custom controls that are derived from standard Swing controls.

There are some custom controls, however, that do not extend the obvious Swing controls. (For example, a customized label may not extend from a JLabel class, rather from the JComponent. The developer could have various reasons for not extending the control from JLabel.) In such cases, iTest will not be able to determine that the JComponent is being used as a label, so the list of actions for the control will not contain label-based capabilities like **getText**. To help in these cases, Spirent supports the **ITestable** interface.

ITestable interface

Developers can implement the ITestable interface in their code to make custom controls more testable. The control supports iTest-related options for controls, for example selecting the name of the label in the component tree or implementing custom actions.

So, in the example that extends from the JComponent, the developer might implement the ITestable interface to let iTest know that the control supports the **getText** action. The developer might also provide a custom implementation of the action. As a result, when iTest analyzes the control, it will add **getText** to the list of actions in the **Actions** tab.

Getting the ITestable interface

The ITestable interface and all supporting classes and interfaces are provided in a jar file called **SwingTestHarness.jar** located in:

<iTestInstallFolder>/plugins/com.fnfr.svt.applications.java.swing.resources_<versionNumber>

Example Code

The following example defines a class called **MyTextPanel** that extends from JPanel. Say I want to use this class to display some text (that is, act as a label). iTest will not know that the custom control acts as a label because it does not extend JLabel. To solve the problem, the custom control implements ITestable to let iTest know that it supports the **getText** action.

Code walkthrough

The **getCapabilities** method identifies for iTest the capabilities of the control (which actions it can perform). We fetch the standard list of capabilities for the control using a static method on **BaseTestable** (**BaseTestable** is provided with the **ITestable** interface). Then we add **getText** to the list of capabilities.

The **isCapabilityImplemented** method asks the control whether it has specific implementation for the action. In the example, we do not want to provide our own implementation of every possible action, but only for the action that we are adding **getText**, so we return true only if the capability is **getText**.

The next method is **performAction**, where the actual action is executed. It returns an IResponse object. In the IResponse interface, you can define what the textual response should be and what should go into the structured data. You can also add images associated with the response. For **getText** however, we want to set only the textual response. We set it to the value of our customized label.

The remaining methods return null. Returning 'null' means that we do not want to provide any implementation for the method and will let iTest use its own algorithm to figure it out. For

example, we do not want to specify the node name of the control (the name as it appears in the tree). We return null to let iTest figure out the node name automatically.

Code

```
public class MyTextPanel extends JPanel implements ITestable {
	/**
	 * get the standard capabilities and add the one you want to the list
	 */
	@Override
	public Collection<String> getCapabilities() {
		Collection<String> result =
BaseTestable.getStandardCapability(this);
		result.add("getText");
		return result;
	}

	/**
	 * Let iTest know that this control knows how to
	 * execute the 'getText' action
	 * This is the only action we are adding to the default list of actions
	 */
	@Override
	public boolean isCapabilityImplemented(String arg) {
		if (arg.equals("getText")) {
			return true;
		}
		return false;
	}

	@Override
	public IResponse performAction(IRequest arg) {
		if (arg.getCapabilityName().equals("getText")) {
			// this is where the code goes to perform the action,
			// in this case to find the response to getText
			final String resultText = "This text is the response to the
getText action";
			// return the result as IResponse
			IResponse response = new IResponse() {
				@Override
				public Collection<String> getErrors() {
					// add errors if you fail to perform this action
					// no errors in this case
					return null;
				}

				@Override
				public Collection<String> getImages() {
					// no images for this action
					return null;
				}

				@Override
				public Document getStructuredResponse() {
					// No structured response to send
					return null;
				}

				@Override
				public String getTextualResponse() {
					return resultText;
```

```
                        }
                    };
                    return response;
                }
                return null;
            }

            /**
             * Return null so iTest uses it's default algorithm
             */
            @Override
            public Collection<ITestable> getChildren() {
                return null;
            }

            @Override
            public String getNodeLabel() {
                return null;
            }

            @Override
            public String getNodeName() {
                return null;
            }

            @Override
            public String getPreferredCapabilityProviderId() {
                return null;
            }

            @Override
            public boolean getProperties(Element arg0) {
                return false;
            }

            @Override
            public Boolean shouldPerformActionInUIThread(String arg0) {
                return null;
            }

            @Override
            public void cancel() {
            }

            @Override
            public Collection<IAttribute> getAttributes() {
                return null;
            }
        }
```

ICapturable interface

iTest can capture actions that you perform directly on the AUT. The **ICapturable** interface enables custom controls to notify iTest when an action should be captured.

In the ITestable interface example, the custom text control was created by extending a JPanel. In the following example, we enable the same control (the **setText** action) to support native capture.

When the control implements ICapturable, iTest will ask it to add a capture listener. Whenever the control needs to notify a new captured action, it will notify the listener with the appropriate capture item.

Sample code

We let iTest know which capabilities should be captured for the control using the **getCaptureCapabilities** method. We can also specify which of those capabilities will be captured by your custom implementation (**isCaptureCapabilityImplemented**).

```
public class MyTextPanel extends JPanel implements ICapturable {
   /**
    * this is a list of listeners to notify a new captures item
    */
   private Vector<ICaptureListener> captureListeners = new
Vector<ICaptureListener>();

   public void addListener(ICaptureListener listener) {
      captureListeners.add(listener);
   }

   public void removeListener(ICaptureListener listener) {
      captureListeners.remove(listener);
   }

   /**
    * Return a list of capabilities that are supported for this component
    */
   public Collection<String> getCaptureCapabilities() {
      ArrayList<String> caps = new ArrayList<String>();
      caps.add("setText");
      return caps;
   }

   /**
    * return if this ICapturable implementation deals with the specified
    * capability
    */
   public boolean isCaptureCapabilityImplemented(String capability) {
      return capability.equals("setText");
   }

   /**
    * this will be called by your code when ever a new setText action needs to
    * be captured
    *
    * @param textValue
    *        - the text that is set
    */
   private void onTextEnteredByUser(String textValue) {
      ICaptureItem item = new SetTextCaptureItem(this, textValue);
      for (ICaptureListener listener : captureListeners) {
         listener.notifyCapture(item);
      }
   }
}
```

Sample ICaptureItem implementation

We have just the **setText** capability, and our sample implementation supports it. We return **true** if it is **setText**, false otherwise.

Now whenever we wish to register a new action, we notify the listeners with the capture item (**ICaptureItem**). In the example, whenever the user enters text, it will call the **onTextEnteredByUser** method, which will create a captured item and notify the listeners.

```
public class SetTextCaptureItem implements ICaptureItem {
     private String capability = "setText";
     private String textValue = "";
     private Component component;
     private long time = 0;

     public SetTextCaptureItem(Component c, String text) {
        this.textValue = text;
        this.component = c;
        this.time = Calendar.getInstance().getTime().getTime();
     }

     /**
      * the name of the action
      */
     public String getCapabilityName() {
        return capability;
     }

     /**
      * time when the action was captured
      */
     public long getCaptureTime() {
        return time;
     }

     /**
      * the command string
      */
     public String getCommand() {
        return textValue;
     }

     /**
      * The component which is the target of the action
      */
     public Component getComponent() {
        return component;
     }

     /**
      * any properties that might be associated with the action. Usually,
      * these need not be specified
      */
     public Document getProperties() {
        return null;
     }

     /**
      * Should iTest mask the command body? This should be true for password
      * fields
      */
     public boolean isCommandMasked() {
        return false;
     }

}
```

Working with JavaApps in iTest Swing sessions

The JAVA Swing session type in iTest allows users to capture actions from a JAVA Swing software application. Those captured actions can be replayed from the capture history or executed from an iTest test case.

How do I know if it is a Java Swing application?

1. Install JDK (if you do not have one) and start **jconsole**. It will show you all processes which are Java processes. If your process is not in the list, it is not a Java process and consequently you cannot using iTest Swing to test it.
2. You can install "procexp" from Microsoft - this is "Task Manager" on steroids. Turn on the loaded dlls" in the lower property pane. If you see "java.dll" loaded, you know that this is most likely a java process.

Swing Session Type Considerations/Requirements

JAVA applications are written in either SWT or SWING. The vast majority of JAVA applications are written in SWING but some are out there that are written in SWT. Applications written in SWT are not supported by this session type.

The user should be running JAVA version 1.5 or later (iTest ships with 1.6).

Java applets (JAVA embedded in web pages) are not officially supported by this session type but there is a way to test certain types of JAVA applets which is covered later in this section.

All JAVA Swing standard objects are supported along with the ability to work with custom objects. Custom objects will cause problems "out of the box" but can sometimes either be worked around by using the **sendKey** action (to manipulate the object with the keyboard). Engineering should be contacted in the event a custom object is encountered.

In order to be prepared to debug any session problem with JAVA swing it is always recommended to turn on debugging. This is set in the iTest preferences. iTest preferences are accessed through the **Window > Preferences** menu item. Locate the **iTest > Session Types > Swing** section. Verify that the **Enable debug mode** checkbox is checked.

To view the console output, open a Console view and select the **swingAppConsole** from the **Display Select Console** icon pull-down:

Session Profile Setup

The first major challenge for any JAVA Swing user will be getting initially connected to their application. Just filling out the session profile is not enough to make iTest work.

Launching A JAR File Or JAVA Class

If you are going to launch a JAR file directly or a specific JAVA class then your session profile will contain the following fields:

Location of the jar file to launch:

> Enter the path and filename of the application to launch (.jar file).

Class to launch:

> If launching a java .class, specify the class name (not the filename).

Classpath:

If your jar/class needs to specify the classpath to run, enter it here.

Working folder:

The folder from which the application under test will be launched from.

Additional command line options:

Specify any command line arguments in which the application needs to run.

Launching A Custom Application

If you are launching a custom JAVA application (.exe) then your session profile will contain the following fields:

Command to launch the application:

Enter the path and filename of the application to launch (.exe file).

Application working folder:

The folder from which the application under test will be launched from.

Additional command line options:

Specify any command line arguments in which the application needs to run.

Connecting To The JAVA Application

In order to allow iTest to "see" and interact with the objects from the JAVA application it is necessary for the user to pass in two JAVA options when called/executing their JAVA application. These options are -javaagent: and -Xbootclasspath/a:.

-javaagent: needs to be the first switch supplied to java. The -Xbootclasspath/a: (/a means 'append') switch(es) can appear anywhere after, but make sure that the customer application doesn't supply a -Xbootclasspath: switch later or it will overwrite the iTest bootclasspath values. -Xbootclasspath/a: arguments can be supplied as semicolon-separated values or multiple -Xbootclasspath/a: can be issued. The -Djavax switch may or may not be needed. The syntax is as follows:

Note Options are entered as a single line, paths are separated by newlines for ease of reading.

```
-javaagent:
"<iTest install
dir>\plugins\com.fnfr.svt.applications.java.swing.resources_<version>\SwingTestHarness.
jar"
-Xbootclasspath/a:
"<iTest install
dir>\plugins\com.fnfr.svt.applications.java.swing.resources_<version>\SwingTestHarnessI
nterfaces.jar";
"<iTest install
dir>\plugins\com.fnfr.svt.applications.java.swing.resources_<version>\SwingTestHarness.
jar";
"<iTest install
dir>\plugins\com.fnfr.svt.applications.java.swing.resources_<version>\xmlwriter-2.2.jar
"
-Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.internal.jaxp.Docu
mentBuilderFactoryImpl
```

When working in Windows, the ^ character can be used to escape a line feed, so a very long command line can appear on multiple lines in a batch file, for example:

```
-Xbootclasspath/a^
:"_<iTest install
dir>_\plugins\com.fnfr.svt.applications.java.swing.resources\__<version>_\SwingTestHarn
essInterfaces.jar"^
;"_<iTest install
dir>_\plugins\com.fnfr.svt.applications.java.swing.resources\__<version>_\SwingTestHarn
ess.jar"^
;"_<iTest install
dir>_\plugins\com.fnfr.svt.applications.java.swing.resources\__<version>_\xmlwriter-2.2
.jar"
```

There are many different types of implementations of JAVA applications. The types outlined below are just examples of application we have encountered so far.

To avoid having to update the launcher file with each new version of iTest, you can add the following:

For Linux:

```
ITEST_HOME=/opt/iTT
SWING_RESOURCE=`\ls -d
$ITEST_HOME/plugins/com.fnfr.svt.applications.java.swing.resources*`
```

Then replace the resources paths with $SWING_RESOURCE, for example:

```
javaagent:"$SWING_RESOURCE/SwingTestHarness.jar"
```

Modify ITEST_HOME= to point to your iTest installation.

JAR File

The best application to use to get to know how to use a Swing session type is the SwingSet2.jar application. This application ships with the JAVA development kit (JDK).

Download the SwingSet2.jar file here: SwingSet2.jar

All that is necessary to launch this application is to browse directly to the JAR file.

LAX File

This type of JAVA application can be identified by locating the directory that contains the applications EXE file. The EXE file will have a file that is named the same but with an LAX file extension. This LAX file is where you can specify JAVA options for the application to use at execution time.

Open the LAX file in Wordpad (or equivalent - but NOT Notepad as there are format issues with that application) and locate the following section:

```
The specific line to edit is the "lax.nl.java.option.additional=-Dswing.metalTheme=steel"
line. In this case there is already a JAVA option specified for this application, the
-Dswing.metalTheme option. The iTest options need to be added as the first options following
the "lax.nl.java.option.additional=" text.
```

Example BEFORE editing LAX file: navdiag_before.lax

Example AFTER editing LAX file: navdiag_after.lax

In order to capture the debug information for this type of JAVA application there is one additional edit that must be made to the LAX file. Located the following section:

Modify the "lax.stdout.redirect=" line to "lax.stdout.redirect=console". This will ensure that all standard out messages will be sent to the iTest console (console view).

Provided all paths and versions are correct the session profile should be able to connect to the application at this point.

Batch File

This type of JAVA application is launched via a batch file. Much like the LAX approach the user must locate the batch file that is being used to execute the application. The iTest JAVA options are added into the batch file.

Example BEFORE editing batch file: lsmsnavigator_before.bat

Example AFTER editing batch file: lsmsnavigator_after.bat

Again, the iTest JAVA options are added as the first options entered.

Webstart

Webstart is unique in that you need to do the same as with a batch file but the batch file is downloaded from the network to a temporary directory when the JAVA application is launched. So, if you try to locate the batch file you will have a very difficult time (unless you know that it does not exist until the application is launched and that it is probably not located in the installation directory). In this case just copy the batch file, make the changes and then launch the batch file directly from the iTest session profile.

Check out Preet's post regarding working with Java webstart apps.

JAVA Applet

JAVA applets are not officially supported by Spirent but for some applets we can use a Swing session. The workaround is to leverage the AppletViewer application that is included in the JAVA development kit (JDK). This application allows users to test their JAR files without having to add the code around the JAR file that is necessary to provide the “frame” in which it will execute. For JAVA applets the “frame” for execution is the web browser but since this does not exist for a Swing session we need to leverage the AppletViewer.

Download the AppletViewer application here: appletviewer.exe

The SwingSet2.jar application from the JAR file section above is also available as an applet on the web. In order to pass the JAVA options to a JAVA applet via the AppletViewer application a batch file is needed. Note that this batch file is differently formatted than the application type batch files referenced above. The same options are necessary to pass in but the method is different as follows:

JAVA options being passed to the AppletViewer application must be prefixed with "-J".

Double-quotes must go around the entire string entry for the Xbootclasspath/a: JAVA option as opposed to each individual path entry.

An example appears below.

Download an example batch file for executing the AppletViewer application to load the SwingSet2 applet: applet_swingset.bat

Note that applets may take a while to load since they are downloaded from the web when the AppletViewer application is launched. This means that you should wait a little while (perhaps several minutes) for the applet to load. It may also be necessary to modify the Swing session profile properties so that the session does not time out while waiting for the applet to load.

Java Swing Object Interaction

The following table summarizes the key ways to interact with Java Swing object in the iTest control panel.

Java Swing Object	**Selection object** (usually the object to be operated on)	**Control Action**	**Action Notes**	**Extraction Action**	**Property for Content/State** (query property("*property*") or XPATH //*property*)
button		click		Describe	Selected
toggle Button		click		Describe	Selected
checkbox		click		Describe	Selected
radioButton		click		Describe	Selected
textField		setText		getText	Text
comboBox		select	Double quote as needed.	getItems Describe	(Response view) SelectedItem
list		select	Double quote as needed.	getItems Describe	(Response view) SelectedValue
ScrollBar		scroll		Describe	Value
slider		scroll		Describe	Value
ProgressBar				Describe	Value
table		doubleClickCell	(to operate on the cell, navigate the tree below the *table* node)	gettable	(Response view)
tabbedPane	One of the tabs	select	Double quote as needed.	getItems	(Response view)
splitPane		mouseDown mouseMove, Use relative xy positioning	May not be possible to see effect while capturing; can only been seen in testcase.	Describe	X Y
TreeDemo_1	tree pane; note that tree elements do not have Java Tree entries	selectNode	Slashes "/" to separate tree elements	getTree describeNode Describe	(Response view) Text UserObject (multiple)

menuBar		menu sendkeys	menu: Slashes "/" to separate menu elements. Double quote as needed.		
menu		contextMenu sendkeys	menu: Slashes "/" to separate menu elements. Double quote as needed.		
sun_swingFilePane_3		TODO		TODO	
editorPane		TODO		TODO	

Java Swing Actions

The Send Key Action (sendkey)

The Java Swing **sendkey** action follows the Microsoft SendKeys Method syntax which is available at http://msdn.microsoft.com/en-us/library/8c6yea83(VS.85).aspx. A summary of their use in Java Swing automation is provided below.

Most alphanumeric and punctuation keys are simply entered without delimiters. The exceptions are the space, +, ^, %, ~, [], and {} must be enclosed in braces, {} such as {+}. Other keys have special patterns, as shown below. If multiple keys are to be pressed simultaneously, enclose them in parentheses, for example +(ab) to indicate shift+ 'a' and 'b' keys all depressed.

Key	Argument
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}

SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1} ...

Key	Special Character
SHIFT	+
CTRL	^
ALT	%

Creating Java Swing Test Cases

Target and Command fields

For Java Swing steps, the **Description** cell specifies two properties: **Target** and **Command**. The properties appear in the **Description** cell in the following format:

```
:target; command
```

Each of the **Target** and **Command** properties has an associated setting for **command, variable, and backslash substitution**. By default, the target has substitution disabled and the command has substitution enabled. This is particularly important keep in mind when performing an iTest variable substitution in the target XPath.

Example

```
://menu[@text = 'File']; left
```

To replace `File` with an iTest variable `i`:

1. In the **Step Properties** for the step, for the **Target** property, check **Perform command, variable, and backslash substitution**
2. In the **Description** text, replace the text with the variable value (`$i`). Escape the XPath items that should not to be interpreted as iTest elements. In this example, we escape the square bracket characters `[]`:

   ```
   ://menu\[@text = '$i'\]; left
   ```

Java applets: Automated testing

Officially, iTest does not support testing Java applets, but you can use a Swing session to test applets. For more information, search for “Java Applet” on the Spirent Knowledge Base.

CHAPTER 57

Syslog Sessions

Syslog session window

In the Syslog session window, you can use a syslog utility to review and/or wait for certain syslog messages. Each Syslog session monitors the syslog messages that arrive at the built-in iTest syslog server (visible in the Syslog view). Only single-line messages are supported.

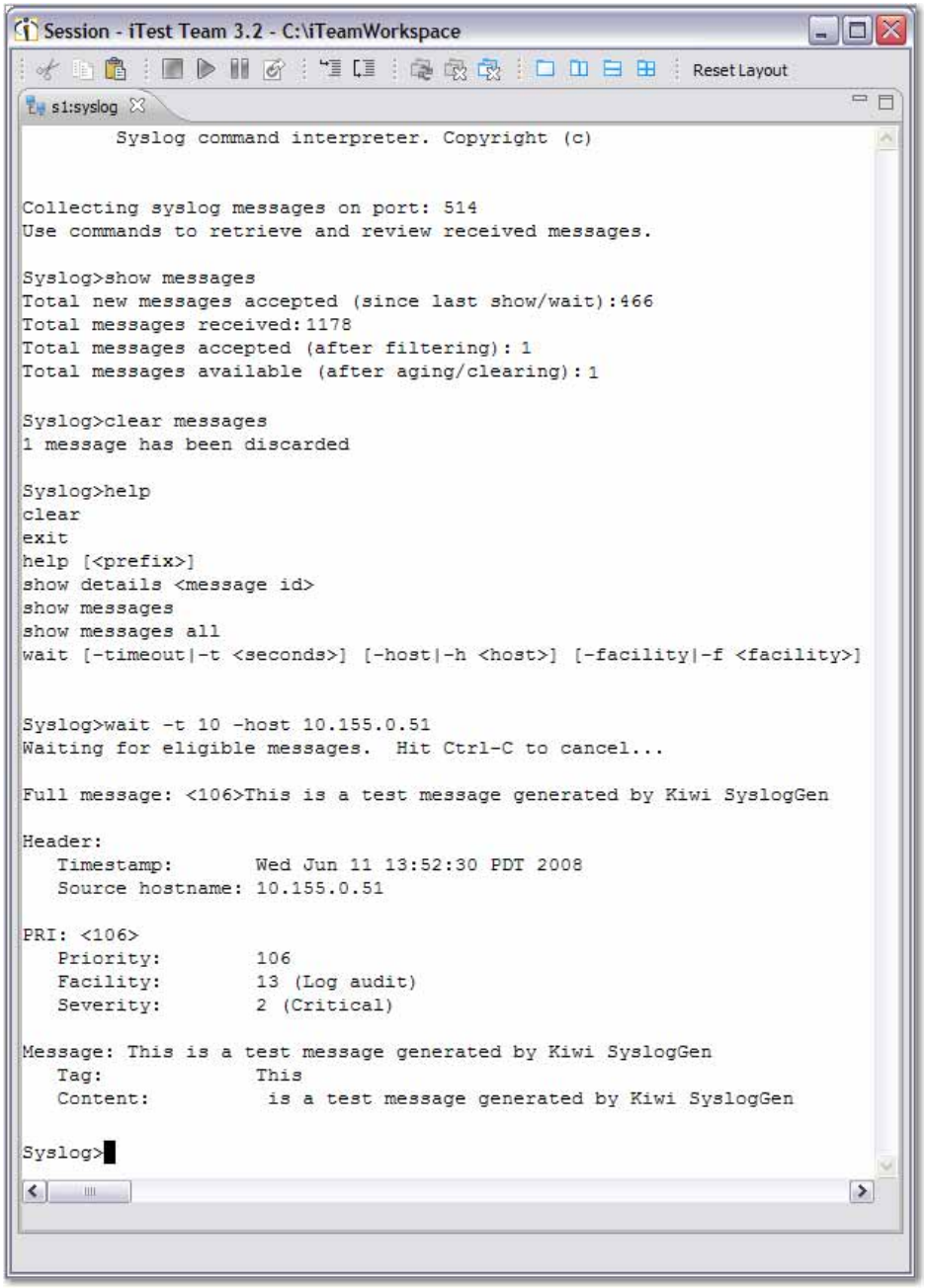
While the syslog server receives all messages from all network interfaces on its specified ports, any syslog session can filter the messages based on the following property settings in the session profile:

- Hostname of the originating computer
- Facility number
- Severity level
- Tag (process): In messages that conform to RFC3164, the first word in the message body is called the “tag”.

As a result of configuring one or more of the settings, only the messages that meet the filter settings appear in the session window. This enables your test cases to analyze the particular responses (messages) of interest and to ignore irrelevant messages.

In addition, in the session profile, you can specify a timeout for responses to **wait** commands.

See “Session profile property settings for Syslog sessions” on page 1141.



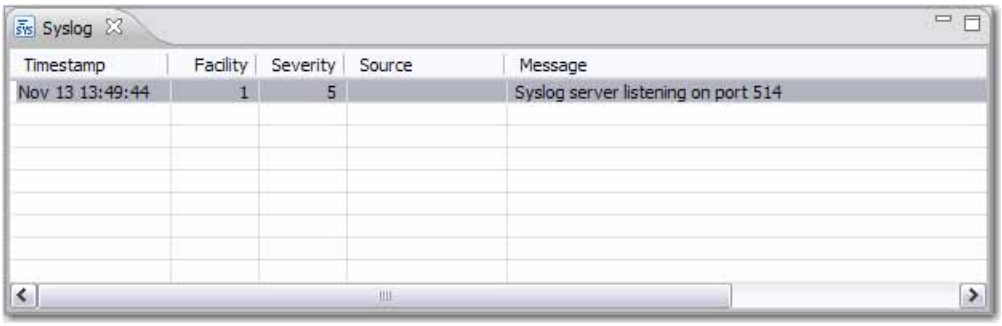
Syslog view

The Syslog view is a read-only display for iTest's built-in syslog server (often called **syslogd**). An instance of iTest supports one syslog server. The syslog server supports single-line messages only.

The syslog server supports multiple syslog receive ports. You can configure ports and other syslog server settings on the **Preferences** page (**Window > Preferences**, then navigate to **Session Types > Syslog**). See "Setting preferences for Syslog sessions" on page 1145 and "Setting iTest preferences" on page 861.

You can learn more about syslog at **http://en.wikipedia.org/wiki/Syslog** and RFC 3164.

To submit syslog commands and analyze responses, open a syslog session. See “Session profile property settings for Syslog sessions” on page 1141 and “Syslog session window” on page 1137.



To view the syslog

Click Show View and select **iTest > Syslog**.

Syslog command set

This topic describes all Syslog commands that you can perform manually in an interactive Syslog session and that a iTest test case can execute.

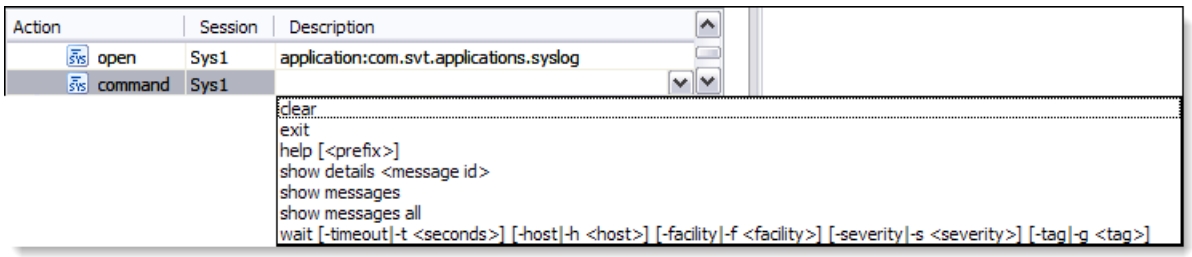
iTest saves all responses to commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or apply analysis rules to the values of interest in the response.

While working in an interactive session

- To view the list of commands while working in an interactive Syslog session, type **help** at the prompt.
- To view detailed help about syntax and arguments, type the command string followed by **?**. For example, to view help on the **wait** command, type **wait ?**

While working on a test case in the Test Case editor

1. For a step in an Syslog session, in the **Action** cell, select **command**.
2. Select the command from the drop-down list in the **Description** cell.



Filtered messages

In the responses to commands, *filtered* messages are messages that were not displayed in the session because they did not meet one of the requirements specified in the session profile settings mentioned in “Syslog session window” on page 1137 (for example, a message that did not come from one of the specified originating computers).

Specifying the ports to listen to

The ports that you specify on the **Preferences** page are the ports that the syslog server always listens to, regardless of whether Syslog sessions are running. The Syslog server listens on all network interfaces on the specified ports (port 514 by default). See “Setting preferences for Syslog sessions” on page 1145.

The ports that you specify in a Syslog session profile determine additional ports for any session that started from the session profile. For example, if the preference setting is 514, and the session profile specifies port 600, then, when the session starts, the syslog server listens on both ports 514 and 600.

Command reference

Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

You can use the following commands when defining steps in a test case:

clear	Clear
exit	Exit, and then close the Syslog session
help [*prefix*]	Display all command syntax and descriptions Specify the *prefix* option to display the prefix for the message
show details *messageID*	For the ports specified as described in “Specifying the ports to listen to” on page 1140: Display details about the message

show messages [**all**]	For the ports specified as described in “Specifying the ports to listen to” on page 1140: Display a summary of messages received. Use the **all** option to display a list of all messages received.
wait [**-timeout**|**-t** *seconds*] [**-host**|**-h** *host*] [**-facility**|**-f** *facility*] [**-severity**|**-s** *severity*] [**-tag**|**-g** *tag*]	Specify the timeout

Session profile property settings for Syslog sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Syslog

Each Syslog session monitors the syslog messages that arrive at the built-in iTest syslog server (visible in the Syslog view). While the syslog server receives all messages, any syslog session can filter the messages based on the following property settings in the session profile.

As a result of configuring one or more of the settings, only the messages that meet the filter settings appear in the session window. This enables your test cases to analyze the particular responses (messages) of interest and to ignore irrelevant messages.

Syslog ports	Optional: Specify a comma-separated list of ports for the syslog server to listen on for this session. This property affects only sessions that use this profile. The syslog server always accepts messages over the ports that you specify on the **Preferences** page. The ports that you specify here determine additional ports for any session that started from the session profile. For example, if the preference setting is 514, and the **Syslog ports** property specifies port 600, then, when the session starts, the syslog server listens on both ports 514 and 600. Default: 514
Max # of messages to keep before aging	Specify the maximum count of messages to list in the SNMP Traps view. When the number of messages reaches the limit, the oldest messages are deleted. Default: 250
Default timeout for wait command	Specify how long to wait (seconds) for the response to a **wait** command. Zero (0) means wait 'forever”. Default: 30 seconds

List of hostnames to accept	Optional. Specify one or more hosts in a comma-separated list. Only messages from specified hosts will appear in the resulting session window. **Note** This property affects only sessions that use this profile. The syslog server always accepts messages from any host on its specified ports. Default: <blank>, which means that messages from any host are accepted.
List of facility #'s to accept	Optional. Specify one or more facility numbers in a comma-separated list. Only messages with specified facility numbers will appear in the resulting session window. This property affects only sessions that use this profile. The syslog server always accepts messages with any facility number. Default: <blank>, which means that messages with any facility number are accepted.
Minimum severity #	Optional. Specify the minimum severity. Only messages with specified facility numbers will appear in the resulting session window. Smaller severity numbers represent higher severity. Therefore, messages with severity values at the specified level and numerically lower will be included while those with numerically higher values will be excluded. **Note** This property affects only sessions that use this profile. The syslog server always accepts messages of any severity. Default: <blank>, which means that messages of any severity are accepted.
List of tags to accept	Optional. Specify one or more tags in a comma-separated list. Only messages with the specified tags will appear in the resulting session window. **Note** This property affects only sessions that use this profile. The syslog server always accepts messages with any tag on its specified ports. Default: <blank>, which means that messages with any tag are accepted.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Setting preferences for Syslog sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Syslog.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Syslog

Comma-separated list of syslog ports	The syslog server receives all messages from all network interfaces on the ports that you specify here. Separate ports using commas. Changes to this setting take effect immediately, with the following exception: Currently active Syslog sessions lose the additional ports that are specified in the session profile (unless they are now specified in this property setting). To activate the ports specified in the session profile, you must restart the sessions. Default: 514 **Note** For some platforms (such as Linux and Solaris), port numbers in this range may be disallowed for use by normal processes like iTest. In these cases, use a different port number (above 1024 in most cases). You must also configure Syslog clients to send to this new port.
Maximum number of messages in Syslog view before aging	The syslog server holds a large number of messages, up to the specified limit. When the message count exceeds the limit, the syslog server deletes (ages) the oldest messages to make room for new messages. Default: 400

CHAPTER 58

Tcl Shell Sessions

Tcl Shell session window

You can type Tcl commands and expressions into the iTest Tcl Shell session window. iTest captures your commands and the interpreter's responses.

See “Session profile property settings for Tcl Shell sessions” on page 1150.

About the Tcl interpreter that iTest uses

The Tcl interpreter provided with iTest can execute third-party Tcl packages that are pure Tcl (no separate Tcl distribution is required, however).

By default, iTest selects the interpreter using the following process:

1. If an interpreter is specified in the **Use the specified Tcl interpreter** property on the preferences page, then use that interpreter.
2. Otherwise, launch the first installed Tcl interpreter that iTest finds in the **PATH** environment variable.
3. If no interpreter is specified in the **PATH** variable, use iTest's built-in interpreter. The internal interpreter is a JACL Java-based Tcl interpreter. Because JACL does not support any C/C++ extensions, most traffic generator devices will not work in this interpreter.

Preferences for the Tcl interpreter are described in “Setting preferences for Tcl Shell sessions” on page 1154 and “Setting iTest preferences” on page 861.

Guidelines

- Commands can span multiple lines
- You can make use of command-line editing and history using the up/down arrows

Sourcing the .itesttclshrc file upon session startup

Before starting a iTest Tcl Shell session, the iTest Tcl Shell interpreter sources the **.itesttclshrc** script located in your home directory (if present). This allows you to initialize the Tcl shell with any startup scripts listed in itesttclshrc. For most .itesttclshrc scripts, the result is to source the standard **.tclshrc** script located in your home directory (if present).

Because the interpreter sources the `.itesttclshrc` file, you can use **[tcl ...]** field replacements in the text of session profile property settings to source Tcl initialization code in the script (which can, in turn, affect the resulting value of the substitution).

Variables

iTest variables are not the same as the variables in a Tcl Shell session. iTest has its own separate data model. If you start multiple Tcl Shell sessions in a test case, then each Tcl Shell session will have variables that are independent of and separate from the variables in other sessions. This isolation of data models has several benefits. For example, this makes it possible to deal with multiple traffic generators in the same test case that would not otherwise be able to co-exist in the same Tcl interpreter because of naming or library conflicts.

Invoking a TCL routine in a test case

Insert a step that opens a Tcl Shell session. You can execute any TCL commands in that session, including sourcing and executing scripts.

Tips for interactive sessions

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.

- During an interactive session, click 

in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Creating Tcl test case steps

Variables

iTest variables are not the same as the variables in a Tcl Shell session. iTest has its own separate data model. If you start multiple Tcl Shell sessions in a test case, then each Tcl Shell session will have variables that are independent of and separate from the variables in other sessions. This isolation of data models has several benefits. For example, this makes it possible to deal with multiple traffic generators in the same test case that would not otherwise be able to co-exist in the same Tcl interpreter because of naming or library conflicts.

Invoking a TCL routine in a test case

Insert a step that opens a Tcl Shell session. You can execute any TCL commands in that session, including sourcing and executing scripts.

Moving data from iTest into a Tcl interpreter session

Moving data is done the same way that you do it interactively in a Tcl shell. To set a Tcl variable to a value, you send a **set** command to a Tcl Shell session with the value coming from iTest – perhaps from a iTest variable.

You will have to pay attention to substitution. By default, iTest substitution of the **Command** property is turned off for steps in Tcl Shell sessions – because otherwise iTest would keep trying to substitute commands and variables that are really intended to be interpreted by the Tcl session.

However, if you turn on substitution on certain steps, you can see how this will work. For example, if the Command **set a $var_1** is placed in the command of a Tcl Shell step, then **$var_1** will be interpreted as a iTest variable if command substitution is turned on, or by the Tcl shell if iTest substitution is turned off. In most cases, then, you turn on command substitution for steps where you are passing iTest variables to the Tcl shell. Don't forget, though, that in these cases, substitution will happen first by iTest and then again by the Tcl interpreter. So you need to be careful about quotes and braces. For example if **$var_1** is a iTest variable that contains the string **hello world**, then using **set a $var_1** will not be correct, because the first substitution by iTest will result in a statement **set a hello world**, which is invalid. Instead, you must add quotes: **set a "$var_1"**. When substituted by iTest, this becomes **set a "hello world"**, just as you want.

Moving data back from the Tcl interpreter to iTest

Responses to Tcl Shell session steps have three parts:

- Result of the statement(s) evaluated
- Text (if any) output to STDOUT while evaluating those statements
- Text (if any) output to STDERR while evaluating those statements.

The parts are combined in the text response body of the iTest step. But they can also be independently accessed using queries on the response. For example, you could add an analysis rule on a Tcl Shell step that stores the result (using the query extractor configured with **result()**) in a iTest variable that you name. Depending on the situation, you might also want to create a iTest response map if the Tcl Shell statement produces a lot of textual output from which you want to harvest specific data.

Session profile property settings for Tcl Shell sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Tcl

Initialization script	Optional. Specify a script to evaluate upon starting the session.

Tcl Interpreter

Use Global Tcl interpreter during execution	Rather than start a new interpreter for the session, use the kernel context Tcl interpreter. Default: Unchecked
Path to Tcl interpreter	Use this option only if your application must use a particular interpreter. Type the directory path to the Tcl interpreter executable. Default: <None>
Path to Tcl Library	Type the directory path to the Tcl library used by the interpreter. To use additional libraries, use the **In addition, use paths specified in the TCLLIBPATH environment variable** property. Default: <None>
In addition, use paths specified in the TCLLIBPATH environment variable	Check the box to make use of libraries specified by **TCLLIBPATH** environment variable. Default: Checked
Initialization script	Optional. Specify a script to evaluate before starting the session with the device.

Large Response

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**

Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Setting preferences for Tcl Shell sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Tcl Shell.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Spirent > Session Types > Tcl Shell

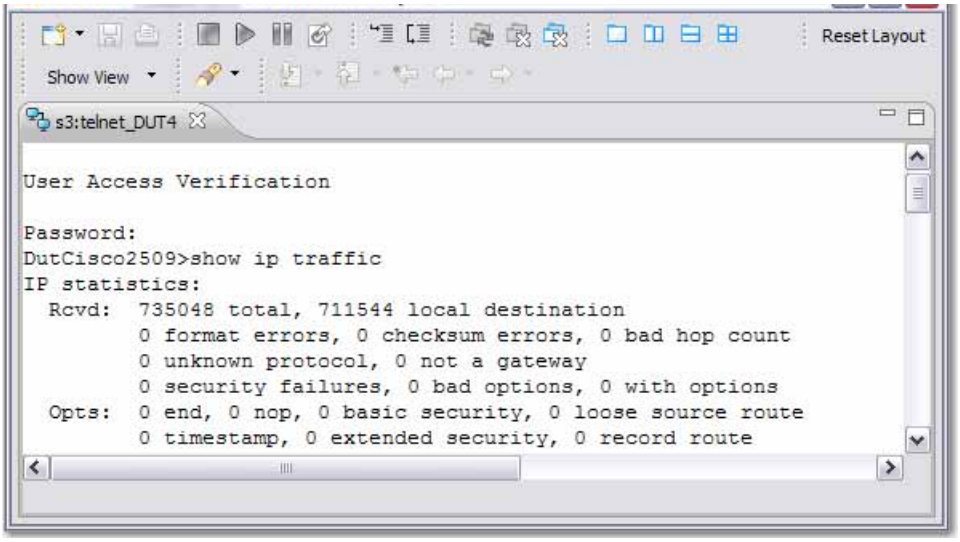
Interpreter	**Note** We recommend that you use the default **Auto-select** setting. **Auto-select**: iTest selects the interpreter using the following process: 1. If an interpreter is specified in the **Use the specified Tcl interpreter** property, then use that interpreter. 2. Otherwise, launch the first installed Tcl interpreter that iTest finds in the PATH environment variable. 3. If no interpreter is specified in the path variable, use iTest's built-in interpreter (base on JACL). **Built-in**: Use iTest's internal JACL Java-based Tcl interpreter. Because JACL does not support any C/C++ extensions, most traffic generator devices will not work in this interpreter. **Use the specified Tcl interpreter**: This option is not often needed. With this option, you specify a particular Tcl interpreter in the text box. Use this option only if your application must use a particular interpreter. Default: **Auto-select**
Log Tcl commands to a console	Check the box to cause iTest to log the submitted commands to a console window. **Note** This setting is used only if you have specified a Tcl interpreter other than the built-in interpreter. Default: unchecked
Log Tcl responses to a console	Check the box to cause iTest to log the responses to Tcl commands to a console window. **Note** This setting is used only if you have specified a Tcl interpreter other than the built-in interpreter. Default: unchecked
Remote shell logging	Use remote shell logging. Default: unchecked

CHAPTER 60

Telnet Sessions

Telnet session window

The Telnet session window displays your commands and the device's responses. You can think of the editor as a terminal client that iTest is monitoring and capturing.



- The Telnet session window looks just like a Telnet client terminal session. When you type a command and press Enter, the editor displays both the command text that you submitted and the device's response.

  The command/response pair, plus some session information is called a ***captured item*** and is listed in the Capture view. Each captured item becomes a step when you save the session as a test case.

- Each session window's tab displays the session type icon and **Session ID** (**S3** in the example) followed by the name of the session profile that was used to start the session (**telnet_DUT4**).

- This example Telnet session with a device shows the login process, a **show ip traffic** command, and the device's response.

- When you close a session (for example, by issuing an **exit** command), iTest captures a **close** Action and then dims the session window.

Tips for interactive sessions

Note Interactive commands are not supported. For example, Start-Service, Stop-Service, Restart-Service, etc., actions that read from console are not supported.

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click QuickCall in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, “QuickCalls: Defining and using a library of custom actions”.

Working with the Microsoft Telnet server

Microsoft’s Telnet server runs in two modes: **stream** or **console** (the default). iTest requires **stream** mode. On Windows XP or Server 2003, type the following commands at a command prompt on the PC that is running the Telnet service:

tlntadmn stop

tlntadmn config mode=stream

tlntadmn start

Session profile property settings for Telnet sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 71. For a detailed description of how iTest uses the settings, see “About property settings” on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Telnet

IP Address or Hostname	Specify the IP address or hostname of the device.
Port	Specify the port for the session. Default: 23

Prompt

For an overview on how iTest recognizes prompts, see "Overview: Prompts in iTest" on page 463.

For instructions on using the properties in this group to define prompts, see "Editing prompt definitions" on page 467.

For related prompt properties, see "Terminal > Replay > Step Defaults > Completion" on page 1209.

Name	The name helps you to remember the type of prompt, for example, **LoginPrompt**. Default: [various]
Content	Specify the exact text of the prompt. **Note**: All prompt definitions are case-insensitive and leading and trailing whitespace is trimmed from any prompt text before iTest attempts to determine whether response text is a prompt. If you use regular expressions in the Content value, then set the Type property to Regex. If the prompt includes a space character or any whitespace in the body of the text, be sure to set the Type property to Wildcard. Default: [none]

Type	Specify the kind of prompt. **Normal**: Interpret the text in the Content field as the case-insensitive text that you expect for the prompt. **Wildcard**: Disregard any characters that appear in the location of the * character in the text specified for the **Content** property. The most common application for the **Wildcard** setting is to allow for leading or trailing numeric or UserID characters in the prompt (for example Device02>, Device03>, and so on). If you set **Type=Wildcard**, then only the * wildcard character is allowed within the **Content** string (and no other wildcard characters like ?). To use other wildcard characters in the **Content** string, you must use **Type=Regex**. **Regex**: Interpret the text specified for the **Content** property as a regular expression. Default: [none]
Is more prompt **More next command** **More quit command**	The **-- more --** prompt is a common method for allowing command line users to view one screen (page) at a time. Many devices use the space character as the command to move to the next page (and often, the letter **q** to exit the display of the response). To enable your automated test cases to page through data that is displayed one page at a time, iTest can automatically “press the space bar” as often as is required to get to the end of the response. As a result, the device's response to the command becomes a single uninterrupted flow of text that does not include the More text. If the prompt is a page-control prompt (for example - - more - -, then: Select the **Is More prompt** checkbox. In the **More next command** text box, specify the command characters (typically a space character) that cause the next page to appear. By default, a space character appears in the box. In the **More quit command** text box, specify the command that exits the More display and returns to the command line prompt. By default, a **q** character appears in the box. Specify a value for **Terminal** > **Replay** > **Step Defaults** > **More**.

Session Properties > More

Telnet > Connect

Connect timeout	Specify how long to wait (in seconds) for the session to start. Default: 30 seconds
Retry count	Specify how often to retry the connection when the connection attempt times out. Default: 1
Negotiate Telnet options	Cause the terminal application to negotiate Telnet options with the host. Default: checked
Ignore Telnet options in data stream	Uncheck the box to parse and implement the bytes in the data stream that encode Telnet options. Check the box to ignore the data. If you check the box and also uncheck the **Negotiate Telnet options** property, then the Telnet session is a raw socket client. Default: unchecked

Telnet > High Availability

For details on implementing tests for HA devices, see Chapter 38, “Testing High-Availability (HA) Devices”.

High Availability	Check the box to enable HA operation. (The default setting, unchecked, specifies normal, non-HA operation.)
Additional connections	Specify the IP address and port pair for each redundant node (nodes other than the master node.). This information is used only by the **open** step for a session. The values in the list represent nodes 1, 2, 3, ... n. Use the following format, one node per line: **<IP_or_hostname>:<portnumber>** **Important**: Be sure not to enter the values for node 0 — the master node — those values are specified by the **IP Address** and **Port** properties.

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Keyboard

Backspace	Specify the code that the device interprets as the backspace character. Default: Ctrl-H
Enter	Specify the code that the device interprets as the Enter character. Default: \r\n

Terminal > Size

Terminal Width	The number of characters that the terminal server presents on a single line. **Same as window size with the specified minimum**: Expand the width based on size of the window, but never less than specified for the **Terminal Width** property. **Fixed as specified**: Force the width to be the value specified for **Terminal Width** property. **Same as window size**: Change the width to be the width of the window. Default:1000
Terminal Height	The number of lines that the terminal server presents on a single screen. **Same as window size with the specified minimum**: Expand the height based on size of the window, but never less than specified for the **Terminal Height** property. **Fixed as specified**: Force the height to be the value specified for **Terminal Height** property. **Same as window size**: Change the height to be the height of the window. Default: 10

Terminal > Style

Note For session profiles that will support TL1 devices, see "Configuring sessions and test case steps for TL1 devices" on page 1215.

Style	Specify the expected format of the responses to commands. **Normal**: The responses will be text (either structured or not structured). **TL1**: The responses will use TL1 formatting. Default: Normal

Terminal > Replay > Step Defaults > Command

Send interval between each character	Some devices require a delay between characters in order to receive commands correctly. If required, specify the interval of time to wait before sending each character in a command. Default: 0 milliseconds

Terminal > Replay > Step Defaults > Terminator

Line terminator	The settings enable you to configure non-standard terminator settings for a session's behavior during replay. For example, special commands like **show ?** do not require the user to press Enter on some devices, but other devices do require the user to press Enter. Typically, you do not need to make any changes to this setting. Default: The default setting (**\r**) is carriage return.

Terminal > Replay > Step Defaults > Response

Treat LF as CRLF	Check the box to cause iTest to change "LF" (linefeed) characters that are returned by the session into "CRLF" (carriage return / linefeed). This is helpful for sessions that send line endings as "LF" only (for example, Python shell). Default: unchecked
File to write response to	Specify the URI of a file to write the responses into. You can use field replacements in the text of the URI to allow the test case to set the filename at runtime. For example, **file:subdirectory_name/[param file_to_create]** The full text of the response is written to the file regardless of the **Number of lines to keep** setting. Default: <none> See the following associated properties: **Append response to file** **Response header** **Number of lines to keep** **Write echo to file** **Write prompt to file**
Response header	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: You may want to specify a text string that should appear before each block of response text. For example: **+-+-+-+-+- Next Response Starts Here -+-+-+-+-+** Default: <none>
Number of lines to keep	For very long responses, you might not want to keep all of the response text (as displayed in the Response view for the selected step while working in the Test Report editor or in the Test Case editor). Specify the maximum number of lines to keep for any single response. Specify **0** (zero) to keep all lines. **Note** If you specify a URI in the **Filename to write response to** property, then all lines in the response are written to the file, regardless of this setting. Default: 10,000

Append response to file	If you save responses to a file by specifying a value for the Filename to write response to property, then: Check the box to append each new response to the file specified in the **Filename to write response to** property. Uncheck the box to replace the text of the file specified in the **Filename to write response to** property with the most recent response. As a result, the file will hold only the last response in the session. See the **Filename to write response to** and **Response header** properties. Default: Checked
Write echo to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: Check the box to include any echoed characters in the saved response. Default: Checked
Write prompt to file	If you save responses to a file by specifying a value for the **Filename to write response to** property, then: • Check the box to include the last line of the response in the saved response (in command-line applications, this is typically the prompt after the response). • Uncheck the box to not save the last line of the response to the file (the prompt at the beginning of the response where the command was typed is still saved). Default: checked

Note

Options **Write echo to file** and **Write prompt to file** do not work in capture mode. These options are available for **Replay** mode only.

The example below shows how the commands/responses are echoed in these scenarios.

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options *are selected*:

```
prompt>command
response text
etc
etc
prompt>
```

Write echo to file

- When **Write echo to file** is *not selected*

```
response text
etc
etc
prompt>
```

- When **Write prompt to file** is *not selected*

```
prompt>command
response text
etc
etc
```

- When **Append to file**, **Write echo to file**, and **Write prompt to file** options are *not selected*

```
response text
etc
etc
```

This is because the **Terminal > Replay** options are used to replay the captured steps. That is, replay the steps captured via test case execution or replayed from the Capture View ("Working in the Capture view" on page 103).

Enable large response truncation	Select these options to manage large session responses. When not selected, all the options below are not available for selection • Truncate responses above given number of line. • Enable execution message upon truncation • Enable execution message upon truncation • Write response to disk upon truncation (for Command prompt, SSH, Serial, and Telnet sessions) When selected, after executing a test, the **Execution view** a warning message displays, for example: **The response is truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir. 2 2 main t1 terminal new_testcase.fftc**
Truncate response above the given number of lines	Enter the number of lines to truncate. For example, 10. When you execute a test with this option, you may verify the response in the **Response view**, which displays 10 lines of response along with the message (for example): **### Response has been truncated. See itest-response_YYYYMMDD-HHMMSS(t1)(step-2) in tmp dir ###**
Enable execution message upon truncation	Select to view/verify the message in Execution
Write response to disk upon truncation	Select to save response to disk. **Note** This option is available only for: Command prompt, SSH, Serial, and Telnet sessions When this option is not selected and you execute a test, you may notice that no response file is generated. That is, no files of the format (in the %temp% folder) after execution of commands: **itest-response_YYYYMMDD-HHMMSS(session-profile)XXXXXXXXXXXXXXX.txt**

Terminal > Replay > Step Defaults > More

Pages to fetch	For responses that are longer than be displayed on a single screen, devices often provide a page-control prompt that enables you to view one screen of text at a time (for example **- - more - -**). Specify the number of pages to fetch when the **more** prompt appears (zero means get all pages). If the setting is non-zero, then iTest retrieves that number of pages and then terminates the output from the session's response by sending the command specified for the **More: Quit Command** property. Default: 100
Device does not remove more prompt. Remove more prompt from response.	Some devices do not remove the text of the more prompt from the text of the response. (For these devices, you will see the more prompt remain at the bottom of the page even after you press the continuation character — typically the spacebar.) Check the box to eliminate the more prompt text from the response that is saved by iTest. Default: Unchecked
Use BELL character to detect end of more pages	Some devices do not remove the more prompt from the screen even after you even after you press the continuation or quit character. Such devices often use the audible bell to alert the user that they have reached the end of the response. Check the box to cause iTest to use the bell as its indicator that the response is complete. Default: Checked

Terminal > Replay > Step Defaults > Completion

You use **Completion** settings to define when the execution of a step should be considered complete. The determination of when a step is complete is protocol-specific. Defining "completion" for a step is important because:

- Some steps cannot start until the preceding step is complete.
- For some steps, you might have defined analysis logic to examine the response to determine whether the step succeeded. Analysis of the response can begin only when the step is complete.

Note In addition, all protocols support the notion of a timeout on a step. If the timeout is exceeded, then the step terminates and execution continues even if the protocol thinks that the work is not yet done. Other than the timeout case, it is up to the protocol to determine when the step is complete.

For CLI protocols, you can specify any of several conditions to define when the step is complete, for example, the existence of certain text in the response or the time elapsed after sending the command. The default setting of the **Completion criteria** property is that the step is complete when:

a The session channel is idle for the time specified by the **Idle channel interval** property

and

b The last line of the response matches one of the prompt definitions specified for the session profile or device.

Idle channel interval	This setting helps in cases where you do not know what response to expect and can use a specified idle time (for example, 100 milliseconds) or when you expect no response whatsoever, for example, when talking to a terminal server. Default: 100
Wait for first character before starting idle	Some devices do not respond to a typed command immediately. This setting enables you to ignore the idle time after the last character of the command is echoed and the first character of the actual response is returned. This way, the delay is not misinterpreted as idle channel time for the purpose of determining completion. Default: True
Completion criteria	**Prompt matches AND device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property and last line of the response matches one of the prompt definitions specified for the session profile. **Prompt matches OR device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property **or** the last line of the response matches one of the prompt definitions specified for the session profile. The following processing order occurs: The step is completed once one of the defined prompts is received. If none of the defined prompts is received or no prompt is defined, then the system waits for the specified **Idle channel interval** time (during which the device sends no response data) and then completes the step. **Device has not sent data during the Idle channel interval**: The step is complete when the channel is idle for the time specified by the **Idle channel interval** property. **Completion time has expired:** The step is complete when the time specified by the **Completion time** property has elapsed. If you specify **Completion time has expired**, then the **Idle channel interval** property setting is ignored. **TL1 End of Message**: For session profiles that will support TL1 devices, see "Configuring sessions and test case steps for TL1 devices" on page 1215 Default:**Prompt matches AND device has not sent data during the Idle channel interval**
Completion time	Specify the time interval that must elapse for the step to be complete. To apply this setting during execution, the **Completion citeria** property must be set to **Completion time has expired**.

Where to find prompt	Specify where the prompt in a response normally appears. **Last line** **Last non-empty line** The **Any line** setting is a special case that you can use to detect a change in state for an ongoing response. For example, you can detect a port's connection status based on whether the first character of a ping response is **u** or **s**. Be sure to use wildcard characters as needed in the **Content** property. Default: Last line
Command to send when a step is cancelled	Specify the characters to send when the user cancels step execution. Default: \03 (Ctrl-C)
Capture only the last screen of response text	Use this property when you expect a very large response, but the only data of interest appears at the end of the response text. Check the box to cause iTest to save only the last screen of response text. Default: Unchecked
Unknown Prompts During Automated Execution For an overview on how iTest recognizes prompts, see "Overview: Prompts in iTest" on page 463. For instructions on defining prompts, see "Editing prompt definitions" on page 467.	
Expected maximum Idle channel interval	The time to wait for a prompt during automated execution. When this time is reached, iTest displays a **Learn this prompt** link in the status bar. • If the user clicks the link, then iTest opens the **Learn Prompt** dialog box to enable you to add the prompt definition. • If the user chooses not to click the link and the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues. Default: 5
Extra wait before alerting user	Additional time to wait for a prompt during automated execution after the **Expected maximum Idle channel interval** time has been exceeded. When (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed, then: • A countdown timer in the status bar starts to count down the **Time for user to respond** time period. • iTest displays a **Keep waiting** link in the status bar. • If the user clicks the **Keep waiting** link, then iTest waits for an additional (**Expected maximum Idle channel interval + Extra wait before alerting user**) has elapsed period. • If the waiting period expires, then iTest raises an execution issue that it failed to find a prompt and sets the test result to **Fail**. The test case then continues Default: 15
Time for user to respond	Specify the amount of time in seconds to wait for the user to respond once the status bar displays the **Waiting for prompt** timer. Default: 30

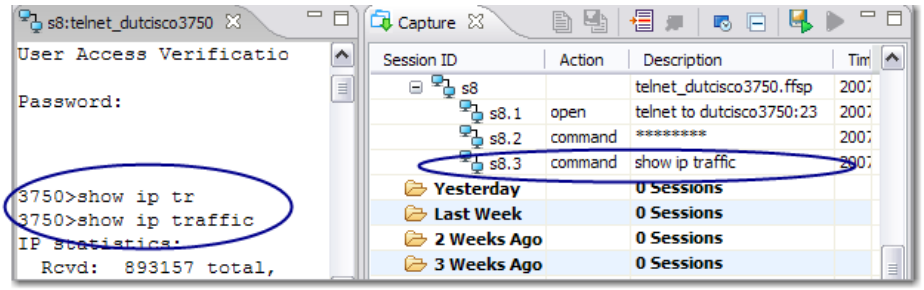
Terminal > Replay > Step Defaults > High Availability

See "Testing HA devices: Detailed instructions" on page 822.

Terminal > Capture

Perform capture cleanup	When you perform manual testing in CLI sessions, you frequently use meta-characters like backspace and up- and down-arrows to correct your typing. In a Capture report, such commands can be difficult to read and understand. If you check **Perform capture cleanup**, then iTest "cleans up" any keyboard shortcuts and removes meta-characters from the captured commands so that the resulting command text appears as if you typed it fully and correctly. See the **Discard command completion steps** property**.** Default: Checked A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, even though we actually typed **show ip tr**<tab>. iTest discarded the intermediate **show ip tr**<tab> form of the command.
Mask unechoed commands	Check **Mask unechoed commands** so that, before creating a Capture report, iTest masks all **Command** property text for which no echo was returned. Check the box to automatically mask passwords. Default: Checked
Remove echo from response	Check **Remove echo from response** to indicate that the device echoes characters typed at the command line. In this case, iTest ignores echoed characters so that the command text is not added to the actual response text. Default: Checked
Remove prompt from response	Check **Remove prompt from response** to save only the response from the session and not the prompt text. We recommend that you do not disable this setting except in rare circumstances. Default: Checked
Use prompts from the session for cleanup	Check the box to use the prompt definitions specified in the session profile document when cleaning up commands. Default: Checked
Discard command completion steps	To ensure that captured commands in the Capture view are easy to understand, iTest, by default, deletes the intermediate command completion text that was submitted while forming a command. A tab completion example appears after this table. iTest captured the **show ip traffic** command correctly, and discarded the intermediate **show ip tr**<tab> form of the command. See the **Perform capture cleanup** property. Default: Checked
Learn prompts	If the box is checked, then, when you close a session and iTest has detected a new prompt, the Update Session Profile wizard starts. You can specify particular prompts in the **Terminal > Prompts** properties. Default: Checked
Learn command completion characters	Learn the character code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can). If the box is checked, then, when you close a session and iTest has detected a new command completion character, t the Update Session Profile wizard starts. The default completion character is tab. To specify particular characters, configure the **Terminal > Capture > Command Completion** property. Default: Checked

Learn break characters	Learn the character code that the device interprets as a break (so you can manually cancel an executing step). The learned break characters are added to the **Command break characters** property. If the box is checked, then, when you close a session and iTest has detected a new break character, the Update Session Profile wizard starts. Default: Checked. The default break character is Ctrl-C. **Note** To specify particular break characters manually, configure the **Command break characters** property (in **Terminal > Capture > Break**).
Remove line containing more prompt	Check the box so that, when capturing responses, iTest deletes the lines that include the more prompt. Default: Checked



Terminal > Capture > Response

Number of lines to keep	Specify the number of lines in the response to keep in the Capture report. Default: 10,000

Terminal > Capture > Command Completion

Specify the code that the device interprets as command completion characters (that is, characters that cause the command interpreter to echo as much of the complete command text as it can).

Default: tab (\t **)**

To determine the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters.

Limitations of the Record feature:

- For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”.
- Function keys are not recorded.

Command completion requires ENTER key	Most devices respond immediately with command completion when they encounter one of the characters specified for the **Command completion characters** property. Set this property to TRUE, if the device, to perform command completion, requires that you press ENTER after typing one of the characters specified for the **Command completion characters** property. Default: Unchecked

Terminal > Capture > Break

Number of lines to keep	Specify the character code that the device interprets as a break (so you can manually cancel an executing step). Default: Ctrl-C To add the encoding for a character set like Ctrl-Z, click **Record** and then press the keys. iTest places the character code into the text box. Click **Add** to add the code to the set of command completion characters. **Limitations of the Record feature:** • For the Alt key, iTest captures only the last key pressed. For example, Alt+q is recorded as “q”. • Function keys are not recorded.

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

CHAPTER 61

TL1 Sessions

Configuring sessions and test case steps for TL1 devices

There are two types of TL1 interface:

- **Automation interface** — These interfaces are not meant for human interaction. These interfaces do not return a prompt and might not even echo what the user types.
- **Hybrid interface** — These interfaces echo what the user types and return a prompt. The prompt can be a normal one like **login:** or **mydut>** or could be TL1 end of message: **<** or **;**

For Automation interfaces, follow this procedure:

- **Configure the testbed device or session profile**

  You will typically work on an Telnet, Serial, or SSH testbed device or session profile.

  1. Set the **Style** property (**Terminal > Style**) to **TL1**.
  2. On the **Terminal > Replay > Step Defaults > Completion** page, set the **Completion criteria** property to **TL1 End of Message**.
  3. No changes are required in the test case.

For Hybrid interfaces, follow this procedure:

- **Configure the testbed device or session profile**

  You will typically work on an Telnet, Serial, or SSH testbed device or session profile.

  1. Set the **Style** property (**Terminal > Style**) to **TL1**.
  2. On the **Terminal > Replay > Step Defaults > Completion** page, set the **Completion criteria** property to **TL1 End of Message**.

- **Configure the test case**

  - For steps that do not return TL1-format responses: Change the completion rule to **Prompt matches AND device has not sent data during the Idle channel interval** (the setting causes iTest to wait for a prompt in the response to the step).

    Note If you see that you would change most steps in the test case to **Prompt matches AND device has not sent data during the Idle channel interval**, you might be better off changing the session profile default to **Prompt matches AND device has not sent data during the Idle channel interval** and changing the steps that have TL1 message bodies so that their completion rule is **TL1 End of Message**.

- For steps that have both a TL1 message body and a prompt: Leave the session profile completion rule unchanged (**TL1 End of Message**). In addition, as long as you have prompts defined in your session profile, iTest will check that the TL1 message is complete and that there is a prompt match.

Actions that provide special settings for TL1 responses

See "The 'readFile' action: Return the contents of a file" on page 256.

Mapping TL1 responses

iTest can typically map a TL1 response without requiring a response map. The TL1 mapper is automatically invoked when the **Terminal > Style** property is set to **TL1** for Telnet, Serial, or SSH devices or session profiles for the step. Tokens extracted using TL1 token mapping are identified by the **<TI1>** XML tag.

There are cases, however, where a response map is necessary or desired to control the queries (and therefore the blue boxes) that are generated for a particular response.

About response mapping in general

Remember that the process of response mapping has two main purposes:

- Parsing an unstructured response and extracting the interesting data into a structured (XML) version of the response.
- Providing meaningful queries to mine the data in that structured portion of the response. The process of mapping is performed either by the session itself (Web, SNMP, all of the traffic generator session types) or by passing a text response (for example, from Telnet, Serial, or SSH) through a response map.

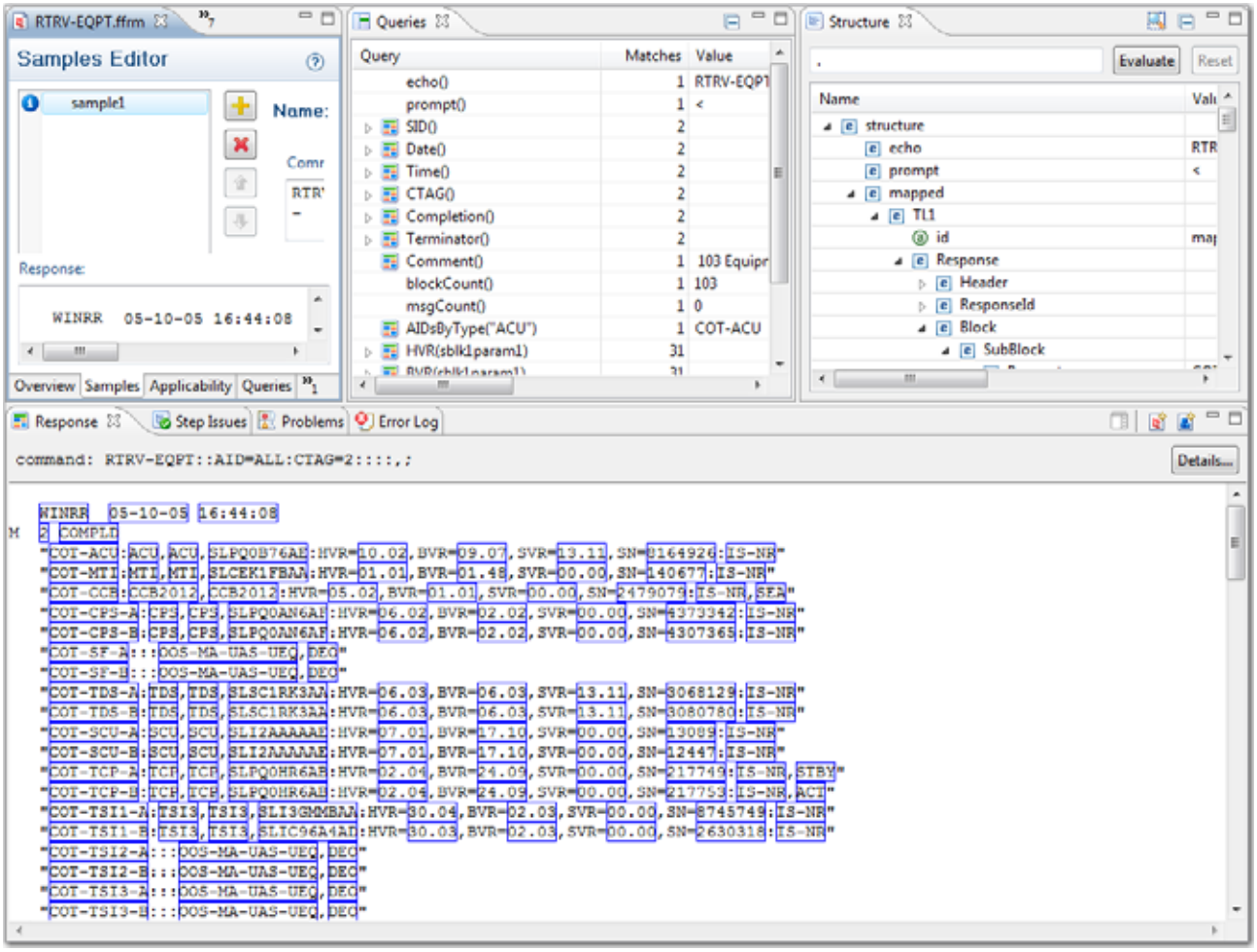
About the TL1 response mapping process

TL1 is a special "in-between" case. The session is typically Telnet, Serial, or SSH, but the response does have a well defined structure — it is just not XML. For this reason, a TL1 response map does not really need any configuration other than selecting **TL1** on the Response Map editor's **Overview** page (there is no need to use table, pattern or block).

A TL1 response map is typically unnecessary because TL1 mapping happens automatically when you set the **Terminal > Style** property to **TL1** for Telnet, Serial, or SSH devices or session profiles.

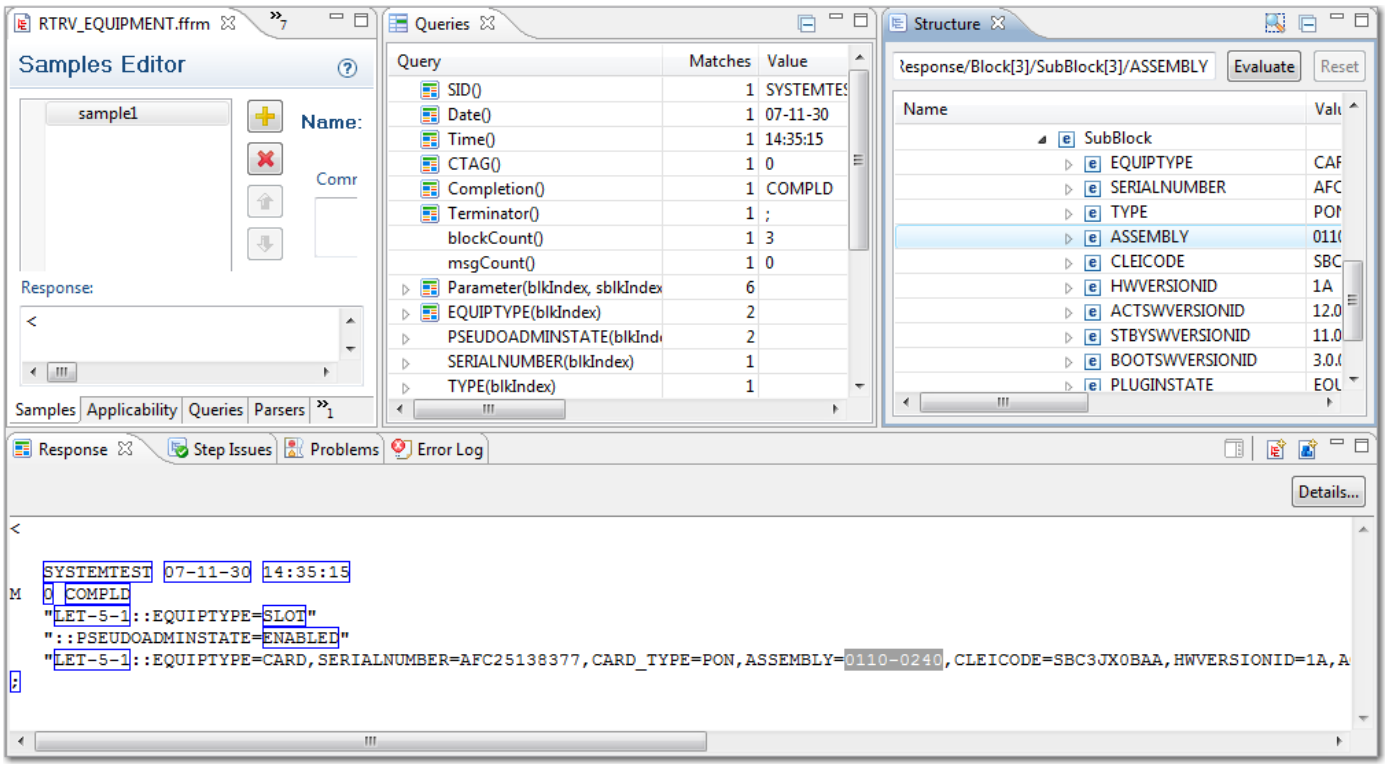
A typical TL1 response looks like this. Notice that almost everything "interesting" is mapped (enclosed in a blue box).

Also notice that many of the queries are keyed by sblk1param1 which is typically known as the AID (COT-ACU or COT-TSI3-B in this example).



Many TL1 responses follow the format in the example where each line is really a row in a table. The TL1 mapper checks whether the first parameter of the first sub-block is unique; if so, it assumes that queries should be keyed by this value. This is not always the case. In this

example, the AID is not unique, so the queries (and therefore the blue boxes) are not really what you would hope for.



There are also other cases where the response may be fully “mapped”, but there are no blue boxes around items of interest. This does not mean that mapping failed. The solution for these cases is to create a “custom query only” response map. That is, a response map whose only purpose is to add or replace the queries that were generated automatically.

Note When creating a “custom query only” map, it is important to launch the **New Response Map** wizard by clicking the **Add Response Map** button on the Response view. If you create a new response map and add a sample (for example of an IxiaTraffic **show stats** response), you may not get the entire response (you will only get the “body” and not the “structure” and “queries”).

Notice that in the example, the Structure view has all of the interesting data from the response. In addition, the value for **ASSEMBLY** is selected in the Structure view and the corresponding text in the Response View is also selected.

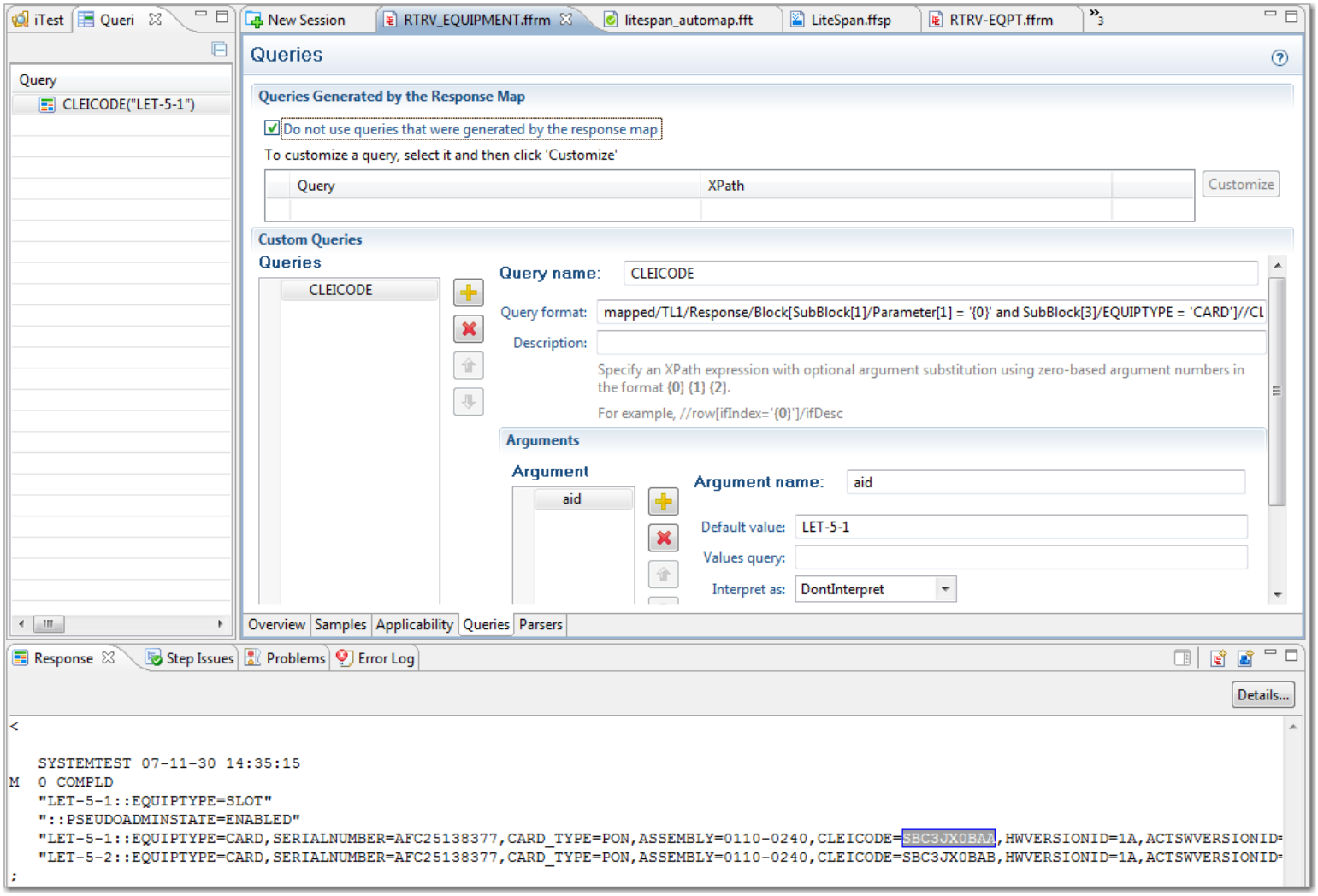
In this example, queries may need to be keyed on both AID and EQUIPTYPE. Let us say you wanted to find the **CLEICODE** value for a particular AID if the AID was of type **PON**. The XPath expression would be:

```
“mapped/TL1/Response/Block[SubBlock[1]/Parameter[1] = 'LET-5-1' and
SubBlock[3]/EQUIPTYPE = 'CARD']//CLEICODE”
```

Tip Use the text box at the top of the Structure view to test XPath expressions.

By turning off the automatically-generated queries and adding a single custom query, you now have full control of where the blue boxes appear and which queries are available for use in analysis rules.

You can further customize the custom query by adding an argument:

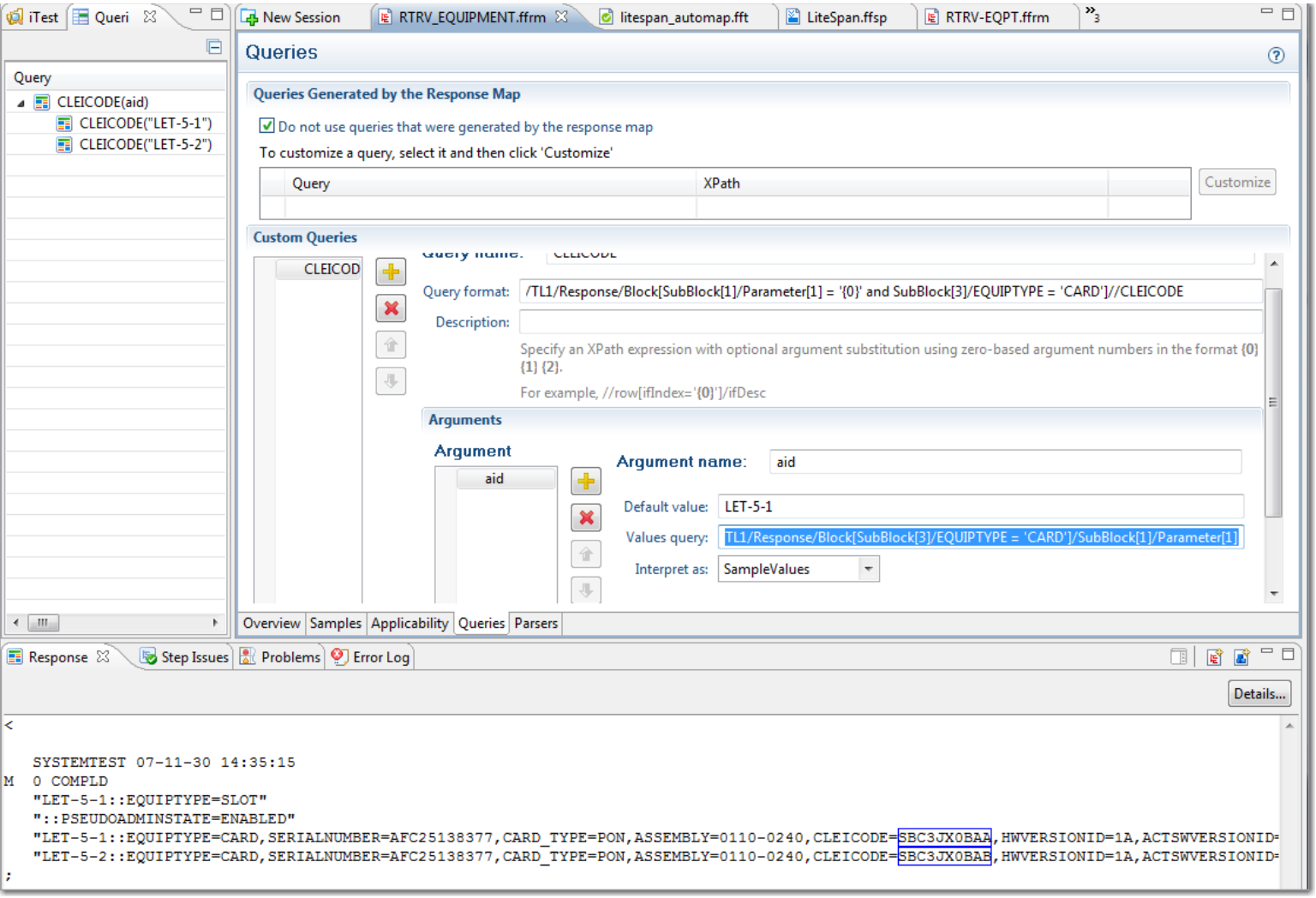


This query would also work for **LET-5-2**, but only the one query/argument instance (based on the **Default value** property) is shown.

Finally, you may add a **Values query**, which is another XPath expression that finds all argument values and results in more queries shown in the Queries view and more blue boxes shown in the Response view. You use a values query only to test your query. The results of the test and verification appear in the Queries view. This is done by parsing values from the values query to your actual query.

The values query for this example is:

```
"mapped/TL1/Response/Block[SubBlock[3]/EQUIPTYPE =
'CARD']/SubBlock[1]/Parameter[1]"
```

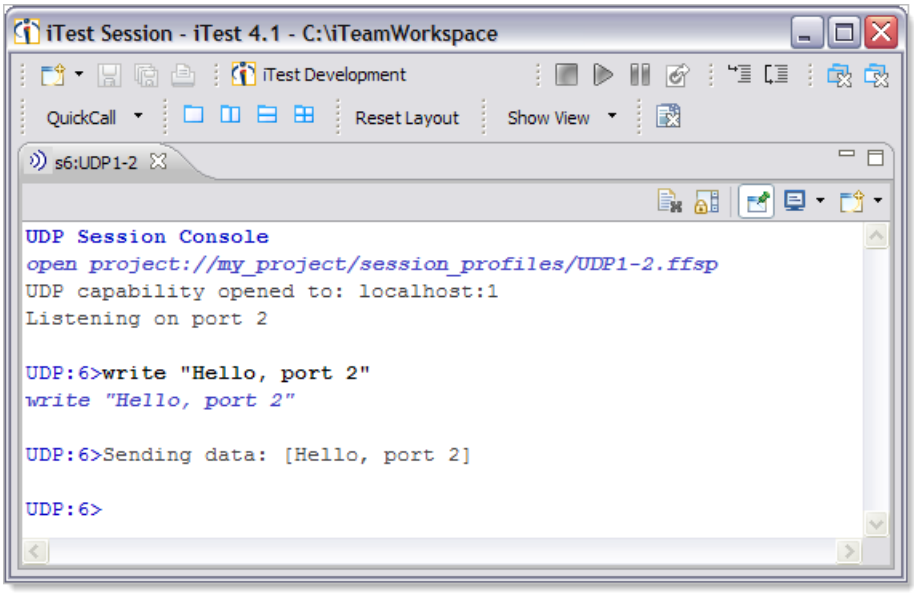


CHAPTER 87

UDP Sessions

UDP session window

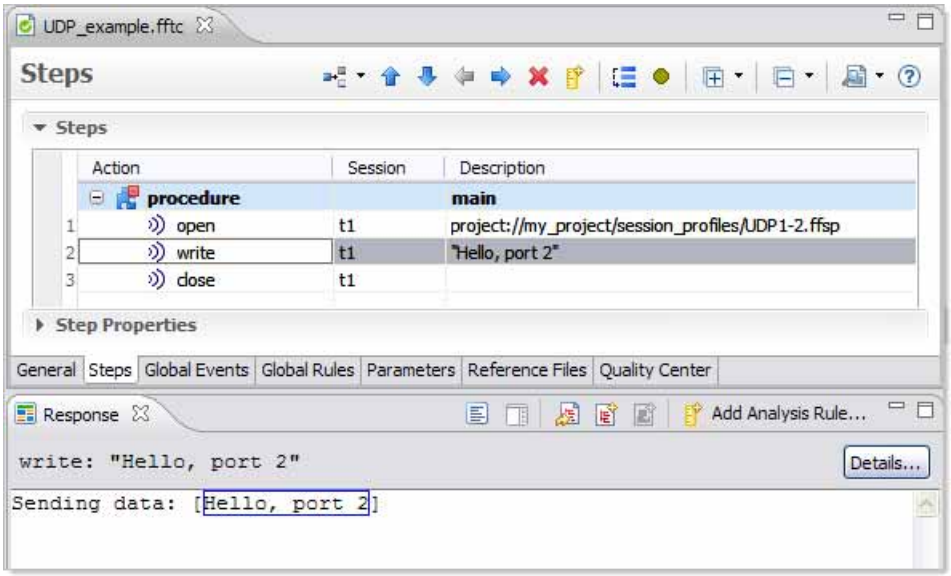
For UDP sessions, the computer running iTest communicates directly over UDP (User Datagram Protocol) with the specified device. For each open session, the UDP session window displays your commands and the local echo. Open another session to view the device's response. You can think of the window as a terminal client — a terminal that iTest is monitoring and capturing.



Responses are mapped

iTest saves all responses to UDP commands as structured data and auto-maps the responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to

create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. Here's the auto-mapped response to a **write** step in a iTest test case.



UDP command reference

Note UDP sessions do not generate iTest Events.

You can send the following commands to UDP sessions:

help	Returns the list of commands and the description of each command. iTest does not map the response to the **help** command.
write "*string1*" "*string2*" ...	Sends the specified strings as the content of the data section of the UDP packet through the UDP socket to the destination port using default system encoding. Use quotation marks and a space to delimit strings. Returns confirmation of text sent. If the **Flush receive buffer when data is sent** session property is enabled, the **write** command also clears the read buffer.
writebytes "*hexString1*" "*hexString2*" ...	Sends the specified data, represented in hexadecimal notation, as raw bytes to the destination. The **writebytes** command bypasses any system encoding. Use quotation marks and a space to delimit strings. Returns confirmation of text sent. If the **Flush receive buffer when data is sent** session property is enabled, the **writebytes** command also clears the read buffer. **Example** **writebytes "1a2b3c4d5e6f"** sends the actual bytes represented by **1a**, **2b**, and so on.

checkbuffer	Checks whether there is data in the read (receive) buffer. Returns true or false.
read <timeout in seconds>	Returns all available data from the listen port. If *timeout* elapses and no data is received, continues to the next step. A *timeout* of 0 (zero) causes the **read** operation to occur immediately.
clear	Clears the contents of the read (receive) buffer. Returns true or false.
close	Closes the UDP session.

Tips for interactive sessions

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click 

in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 10, “QuickCalls: Defining and using a library of custom actions”.

Session profile property settings for UDP sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

UDP Session properties

Destination IP address	IP address or hostname of the remote device.
Source port	Port from which UDP traffic from the session is sourced. This value will appear in the **source port** field of the UDP packet. **Limitation** Many sessions can share the same source port for sending, however, only one session can "listen on" a port to receive responses. This is an OS limitation—only one process can bind to and read from a particular server port at a time. See the **Port to listen on for responses** property. Default: 6334
Destination port	Port to send data to. Default: 1
Port to listen on for responses	Port to use when listening for response data. **Note** The port number is not returned in the response for **read** commands. **Limitation** Many sessions can share the same **Source port** for sending, however, only one session can "listen on" a port to receive responses. This is an operating system limitation—only one process can bind to and read from a particular server port at a time. See the **Source port** property. Default: 1
Flush receive buffer when data is sent	Check the box to cause the **write** and **writebytes** commands clear the read buffer after sending the data. Default: checked

CHAPTER 88

VNC sessions

VNC session window

Important iTest VNC sessions are intended to help you to control a remote OS to perform configuration/setup/tear-down tasks. iTest VNC sessions are not intended to enable you to thoroughly test an application on a remote platform.

iTest captures your VNC session so that you can:

- Interact with a remote GUI environment (primarily Windows) using VNC. As a result, you can effectively use administrative tools to configure the remote environment or execute manual test cases within the context of a standard tool.
- Record and review the actions executed in a remote environment through the VNC protocols. As a result, you can have a record of the actions performed during a test or administrative action and can use those as the foundation for future automation.
- Interact with the command shell on a remote Windows environment so you can apply standard iTest CLI automation technology to the Windows shell (for example, response mapping). As a result, you can use a broader range of iTest features and potentially have a greater degree of control than a standard VNC playback may afford.
- Replay actions previously recorded over a VNC session in a robust manner, independent (to the extent possible) of changes in window position, layering, extra windows or controls on the screen, dynamically changing content, and other variations in the user experience. As a result, your replay scripts are robust and automation is repeatable.

Supported security methods

The VNC protocol is extensible to enable servers to provide several authentication methods. iTest supports two authentication methods:

- None
- VNC Authentication

If you see an error message like “VNC security negotiation error: Unknown security type”, then change the server authentication settings to one of the supported methods.

Connecting iTest from your desktop through a firewall to your lab devices

Using iTest over a VNC connection to try to connect to lab devices through the lab firewall is difficult (wrong screen dimensions, latency, no tooltips, and so on) and can require complex setup and troubleshooting. Instead, use iTest's SSH **Local Port Forwarding** capability for a very

clean connection. See “Using SSH Local Port Forwarding to connect iTest from your desktop through a firewall to your lab devices” on page 1522.

Best practices for automating VNC test cases

Searching for target images is not a reliable way to perform actions in an automated VNC test case. Instead, follow these practices to improve the robustness of test cases:

- Turn off all visual effects that can be turned off
- Turn off font smoothing
- Do not use wallpapers
- Use **sendKeys** when possible
- Maximize windows when possible

Tip On Windows, maximize a resizable window by pressing **Alt+Space+X** (the **“% x” sendKeys** action)

Setting Microsoft Windows 7 firewall for VNC sessions

If you get connection time-outs in iTest VNC sessions, but not when using a native VNC viewer, then your Windows firewall may be blocking the iTest program and its traffic.

- In Windows 7, set two inbound rules: one rule for TCP and one for UDP
- Allow any local/remote address, protocol, port, computer or user access to iTest. Be sure to apply the rules on all profiles (domain, private, public, and so on)
- Set encryption to **always off** or **preferred off**
- Set authentication to either **none** or **VNC password**

VNC session toolbar

	**Take Snapshot**. A “snapshot” is a visual image of the VNC session window. See “snapshot” on page 1636
	**Get Server Clipboard**. performs and captures a **getClipboard** action. Gets all the contents of the server clipboard. See “getClipboard” on page 1634
	**Send Special Keys**. Send and capture the keys that are selected from the drop-down list. See “sendKeys” on page 1635
	Performs and captures a **captureTarget** action — the selected region and its X, Y coordinates are captured to the **Capture Target Step Properties** group as described in “Targets and target properties for VNC steps” on page 1636 **To capture a target** 1. Click the **Target Select** button  2. Select the target (any region) on the screen. 3. Press **Enter**. iTest performs and captures a **captureTarget** action. See “captureTarget” on page 1633

VNC session action reference

The following actions are available in VNC session steps in test cases.

captureTarget

During execution	**captureTarget** waits for a target for the specified amount of time, but no events are sent to the server. **captureTarget** is useful for determining whether an element is present on the screen. You can use **captureTarget** in combination with the **targetFound** response item for test case validation
While capturing a manual session	**In an interactive (manual) session:** 1. Click the **Target Select** button  on the VNC toolbar. 2. Select any region on the screen. 3. Press **Enter**. iTest performs and captures a **captureTarget** action — the selected region and its X, Y coordinates are captured to the **Capture Target Step Properties** group as described in “Targets and target properties for VNC steps” on page 1636. See “VNC session toolbar” on page 1632.
Target	Required
Properties	See “Targets and target properties for VNC steps” on page 1636 and “Responses to actions that use targets” on page 1637.

click

During execution	Moves the mouse to the specified target and then simulates the mouseDown and mouseUp at that location
While capturing a manual session	A click action is captured when a mouseDown and mouseUp happens at the same location or in succession. Key mask properties (Ctrl, Shift, Alt) are captured.
Target	Required
Properties	The **Command** property for the step specifies mouse button (left, middle or right, case-insensitive). If blank, then left-click used. The **Click Step Properties** group contain checkboxes for key modifiers (Ctrl, Alt, Shift). See “Targets and target properties for VNC steps” on page 1636 and “Responses to actions that use targets” on page 1637.

doubleClick

During execution	Performs two **click** actions in quick succession (default: 500ms between clicks). This interval can be picked up from the OS using the Java system property. The **Command** property for the step specifies mouse button (left, middle or right, case-insensitive). If blank, then left-click used.
While capturing a manual session	If two clicks happen on the same target within 500ms, then a **doubleClick** action is captured (not **click)**.
Target	Required
Properties	The **Click Step Properties** group contain checkboxes for key modifiers (Ctrl, Alt, Shift). See “Targets and target properties for VNC steps” on page 1636 and “Responses to actions that use targets” on page 1637.

getClipboard

During execution	The **Get Server Clipboard** button  on the VNC toolbar performs and captures a **getClipboard** action. Gets all the contents of the server clipboard. The text is added to response with query **text()**. If the server does not support clipboard operations, then an empty string is added to the response. See “VNC session toolbar” on page 1632. See “setClipboard” on page 1636.
While capturing a manual session	The **Get Server Clipboard** button  on the toolbar performs and captures a **getClipboard** action. The button is disabled if the server does not support clipboard operations. See “VNC session toolbar” on page 1632.
Target	None
Properties	None

keyDown and keyUp

The **keyDown** and **keyUp** actions simulate keyboard keystrokes — up or down. You can specify only one key in the command using the same notation as described for **sendKeys.**

During execution	Sends the specified keystroke.
While capturing a manual session	Not captured.
Target	None
Properties	None

mouseMove

During execution	Moves the cursor to the specified target and then waits there for the time specified by the **Hover time** property.
While capturing a manual session	Most motion of the cursor is not captured. In contrast, a **mouseMove** action is captured if the cursor moves to a location and stays there for the period specified by the **Hover time** property. Only one mouseMove action is captured for each such hover activity. Key mask properties (Ctrl, Shift, Alt) are captured.
Target	Required
Properties	**Hover time** is the time to hover the cursor over the specified location (default 2 seconds). The **Click Step Properties** group contain checkboxes for key modifiers (Ctrl, Alt, Shift). See “Targets and target properties for VNC steps” on page 1636 and “Responses to actions that use targets” on page 1637.

mouseDown and mouseUp

The **mouseDown** and **mouseUp** actions enable you to simulate advanced behavior, for example, drag and drop. A drag and drop is a combination of **mouseDown**, **mouseMove**, and **mouseUp.**

During execution	Moves the mouse to the specified target and simulates **mouseDown** or **mouseUp**.
While capturing a manual session	Not captured.
Target	Required
Properties	See “Targets and target properties for VNC steps” on page 1636 and “Responses to actions that use targets” on page 1637.

sendKeys

sendKeys simulates pressing a series of keys.

During execution	Sends the sequence of keys specified in the **Command** property. See “Key notation” on page 1635.
While capturing a manual session	As you type, keystrokes are added to a buffer. If no **keyUp** action is captured for 2 seconds, then the keys in the buffer are collectively captured as a single sendKeys action. For example, while typing, you pause for 2 or more seconds and then start typing again — two **sendKeys** actions are captured.
Target	None
Properties	The **Command** property for the step specifies the sequence of keys.

Key notation

Key notation is defined by the MSDN specification:
http://msdn.microsoft.com/en-us/library/8c6yea83%28VS.85%29.aspx

Differences between the iTest sendKeys notation and MSDN notation:

- MSDN specifies aliases {BS} and {BKSP} for {BACKSPACE}. iTest supports only {BACKSPACE}
- MSDN allows one to send multiple keys using **{key n}**. iTest does not support that notation.
- The MSDN document is unclear regarding uppercase characters (like **T**). In iTest, instead of **+t**, you specify **T**. For example, to send **Ctrl+Shift+T**, send **^T**
- Caps Lock is not supported

setClipboard

During execution	Sends a command to the server requesting the contents of the server clipboard. See “getClipboard” on page 1634.
While capturing a manual session	Not captured
Target	None
Properties	None

snapshot

During execution	Takes the image of the VNC session window and adds it to the response body.
While capturing a manual session	The **Snapshot** button on the VNC session toolbar performs and captures a **snapshot** action. See “VNC session toolbar” on page 1632.
Target	None
Properties	None

Targets and target properties for VNC steps

A target is the area on the screen where the action is performed (for example, a **sendKeys** action on a dialog box or a **click** action on a button). Actions are performed at the center of the target image.

During execution, the system can locate a target either by absolute screen coordinates (X,Y) in pixels, or by searching for the image of the target based on comparisons with the captured image. You specify this option using the **Search for image** property setting.

Important Searching for target images is not a reliable way to perform actions in an automated VNC test case. See “Best practices for automating VNC test cases” on page 1631.

◆ To view target properties

Select a step in the Test Case editor. In the **Properties** section, for the **VNC Client <actionName> Properties** group, navigate to **Target**.

Property settings

The following properties are captured for many VNC steps.

X, Y	Specify the absolute **X** and **Y** coordinate of the target (pixels). Default: 0,0
Search for image	Check the box to search for the image on the screen by comparing the screen images with the captured image that is held in the **Base64 image** property setting. Uncheck to use screen coordinates to locate the image. Default: checked
Target wait timeout	Applies only when **Search for image** is checked. Specify the maximum number of milliseconds to wait for a target to appear before performing the action on the target. Default: 2000 (2 seconds)
Use X-Y when image is not found	Applies only when **Search for image** is checked. If checked, and the image search fails after the time specified by **Target wait timeout**, then use the specified X and Y screen coordinates to locate the image and continue execution. Otherwise, generate an execution issue. Default: checked
Base64 image	Encoded image. Base-64 encoded integer array with format: [width, height, relative x, relative y, 32-bit pixels].

Responses to actions that use targets

- Actions that use target properties add the **targetFound** Boolean to the response structure for the step.
- Actions that use target properties can add two images to the response structure for the step:
  - **Actual image**: The image that was returned by the step
  - **Expected image**: The image that is specified for the **Base64 image** property for the target

Note If the step used coordinates to locate the target, then the **Expected image** is not returned in the response.

Session profile property settings for VNC sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

VNC

Host	Required. Specify the IP address or hostname of the host running the VNC server.
Display	Specify the VNC display value (integer) to use (zero-based). While connecting to the server, the VNC client adds the **Display** property value to **Base Port** property value (in the **Advanced** properties) and uses the resulting value as the connection port. This allows a server to run a separate VNC session on each of multiple displays. By default, VNC servers bind to display 0 for Windows and to display 1 for Linux. Default: 0
Password	Optional. Specify the password required to connect to the VNC server .

VNC > Advanced

Base port	Port on the server. Default: 5900
Connection timeout	Specify the maximum time (in seconds) that iTest should wait to establish a connection with the server. If a connection fails, iTest will retry after a short interval until the connection is established or the specified **Connection timeout** has elapsed. Default: 10
Scale to window size	This setting enables you to capture screenshots at the size that is currently displayed on the monitor, as compared to capturing at the screen resolution of the application under test. This is useful, for example, when the resolution of the AUT is very high, resulting in a physically large screenshot at the default scale setting (and as a result, only a part of the screen appears in the snapshot). Check the box to capture the screenshot at the size that it currently appears in the VNC window. Default: unchecked
Show cursor	Check the box to include the cursor icon (the mouse pointer) in the captured image. Default: unchecked

VNC > Heuristics

Click distance	Specify the maximum distance (in pixels) between the actual locatio of the target and the **mouseUp** action that should be captured as a **click** action on the target (that is, how far away can the cursor be from the target). See “click” on page 1633 and “doubleClick” on page 1633. Default: 5
Double-click interval	Specify the maximum duration (in milliseconds) between two clicks that should be captured as a **doubleClick** action. See “doubleClick” on page 1633. Default: 500
Send keys timeout	Specify the maximum duration (in milliseconds) between subsequent keystrokes that will be captured in a single action. See “sendKeys” on page 1635. Default: 2000
Target image width	Specify the default width (in pixels) for targets captured during interactive sessions. See “Targets and target properties for VNC steps” on page 1636. Default: 100
Target image height	Specify the default height (in pixels) for targets captured during interactive sessions. See “Targets and target properties for VNC steps” on page 1636. Default: 100
Send keys interval	Specify the interval (in milliseconds) between generated keystrokes during test case execution. See “sendKeys” on page 1635. Default: 100
Mouse move interval	Specify the interval (in milliseconds) between capturing **mouseMove** actions during interactive sessions. See “mouseMove” on page 1635. Default: 2000

CHAPTER 89

VMware vSphere Client sessions

vSphere sessions in Spirent iTest

The session window for vSphere sessions in iTest has been designed in partnership with VMware to closely resemble the vSphere client. As a result, you can capture vSphere steps using iTest without having to learn a new interface or new command names. You interact with the iTest session almost exactly like you interact with vSphere.

The iTest interface to vSphere enables you to perform and automate a wide range of tasks, including:

- Orchestrate VMs from iTest
  - CRUD support: Create, retrieve, edit, and delete
  - Start/Suspend/Stop VMs
- Create VMs from scratch or clone from VM template
- Manage VM snapshots
  - Take snapshot
  - Revert to snapshot
  - Delete snapshot
- Test virtualized environments
  - Set up VMs
  - Start VMs, login to software, perform test
  - Log out and stop VM as part of test clean up

Limitations

- You cannot start up a VM and then configure it. All VMs must be pre-configured in vSphere.
- Each VM must have a pre-configured IP address

Actions and responses are captured

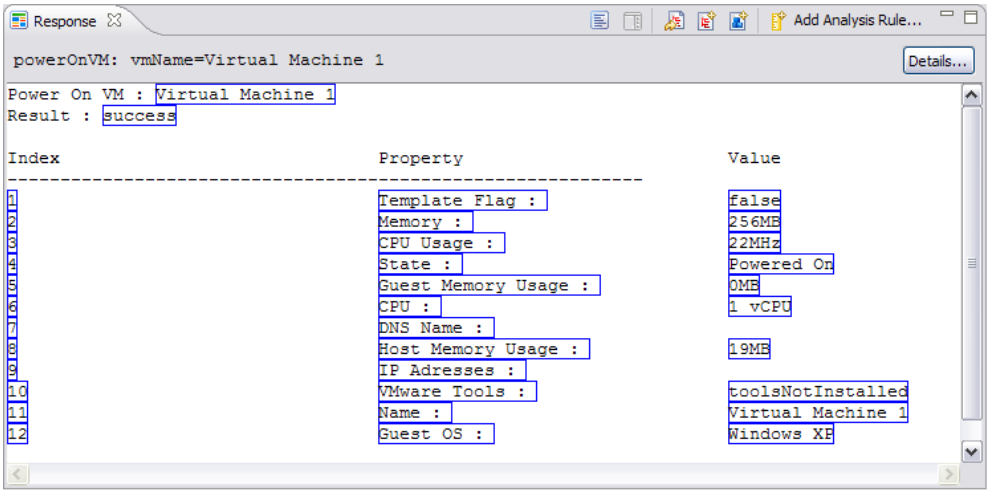
Because all actions and responses are captured, you can use captured items to create test case steps that interact with the vSphere server.

- Any action that you perform in the iTest session is forwarded to the vSphere server. vSphere performs the action and returns its normal response. You can view the response on the iTest window, just like you do in vSphere.

- iTest captures all of the actions that you perform in a vSphere session and all of the responses returned by vSphere.

Responses are auto-mapped

iTest saves all responses to your actions as structured data and then auto-maps the responses (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the auto-generated queries. Here's the auto-mapped response to a **powerOnVM** step in a iTest vSphere session. As you can see, auto-generated queries return all important data tokens for your use in analyzing the response:.



Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Creating a test case from an interactive vSphere session

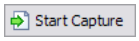
Note This topic provides an example of working in an interactive vSphere session. For more detailed information on working in the session window, see "Managing hosts, datacenters, datastores, resource pools, and VMs" on page 1645.

Typically, you use iTest to generate a virtual testbed that you will use to test your application. A typical iTest test case might:

1. Call an iTest procedure that creates and starts up all of the required VMs in the required configurations.
2. Test your application in the virtual test lab and collect and analyze the data of interest.
3. Call a different iTest procedure that tears down the virtual testbed.

Note In an iTest session with vSphere, the processes of creating and managing elements follows the same flow as in vSphere. If iTest opens a dialog box to allow you to specify settings to perform the task, follow the procedure that you use in vSphere. For more details on any dialog box, see the VMware online help.

Example: Capturing vSphere steps and saving them as an executable procedure

1. Ensure that the vSphere session profile or device is properly configured. See “Session profile property settings for VMware vSphere sessions” on page 1650.
2. Click 

to begin the direct-to-test process of saving the interactive session as a test case.
3. Start the vSphere session in iTest and work with VMs as needed. You work in the iTest vSphere session the same way you work in vSphere. When you interact with a vSphere component, iTest submits the vSphere action to the vSphere server and captures both the submitted action and the response from the server. For example:

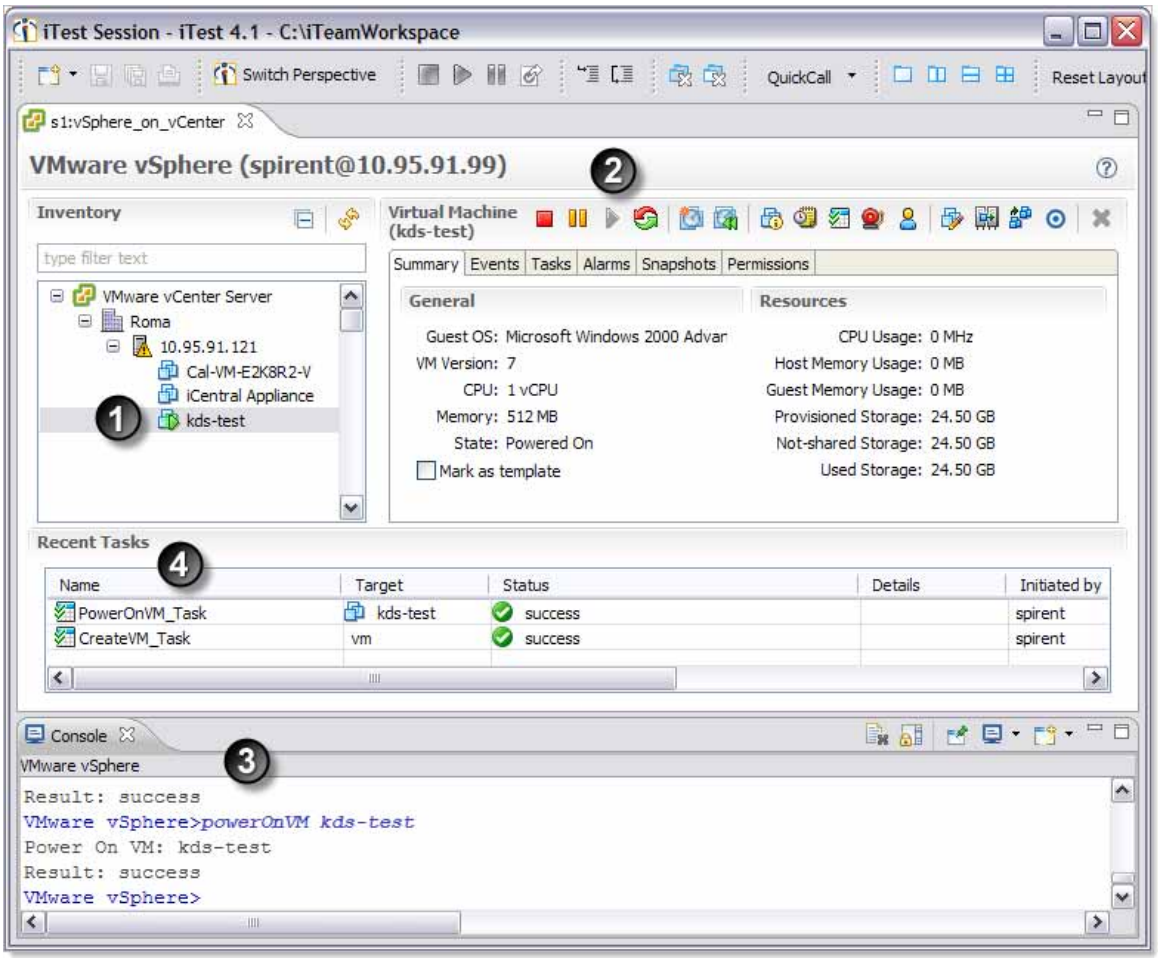
You start the session and iTest connects to the vSphere server. The server returns confirmation data to iTest.

1. You select the VM named **kds-test** in the list of VMs. Notice the decorators on the icons that represent the state of the item (powered on, suspended, error, and so on).

2. You click the button to power on the VM.

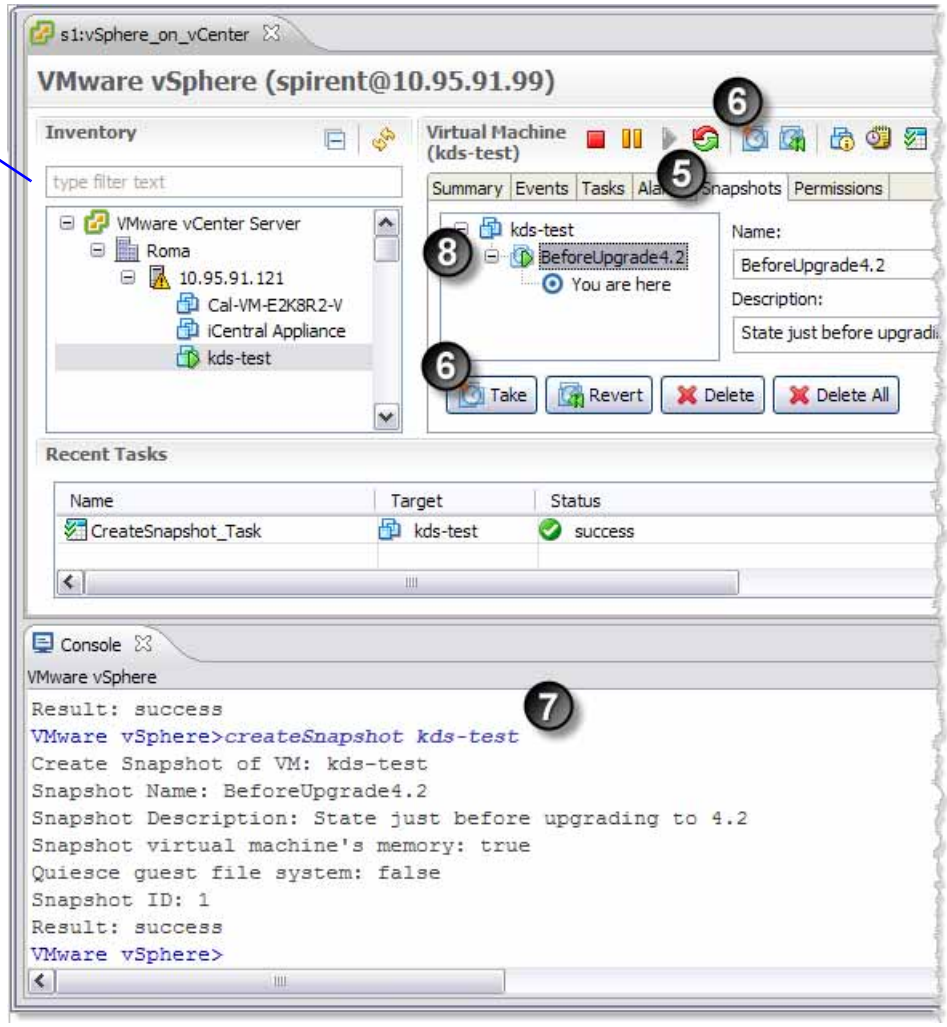
3. iTest submits a **powerOnVM** action and the vSphere server powers the VM on and then returns data about the VM to iTest. iTest displays the server's response in the Console view.

4. The task and its status appear in the **Recent Tasks** pane.



5 Now, let's take a snapshot of the VM: Click the **Snapshots** tab:

Tip: Type text in the filter text box to display only entities that include the text.



6 Click the **Take Snapshot** button (two options—in the main toolbar or on the **Snapshots** tab). You fill in a dialog box with the name and description of the snapshot.

7 iTest submits a **createSnapshot** action and the vSphere server creates the snapshot and returns data about the new snapshot to iTest.

iTest captures a step that includes the command that was sent to the server and the server's response. The captured step is added to the resulting test case, and, when you run the test case, the step will execute exactly as you performed it in the interactive test.

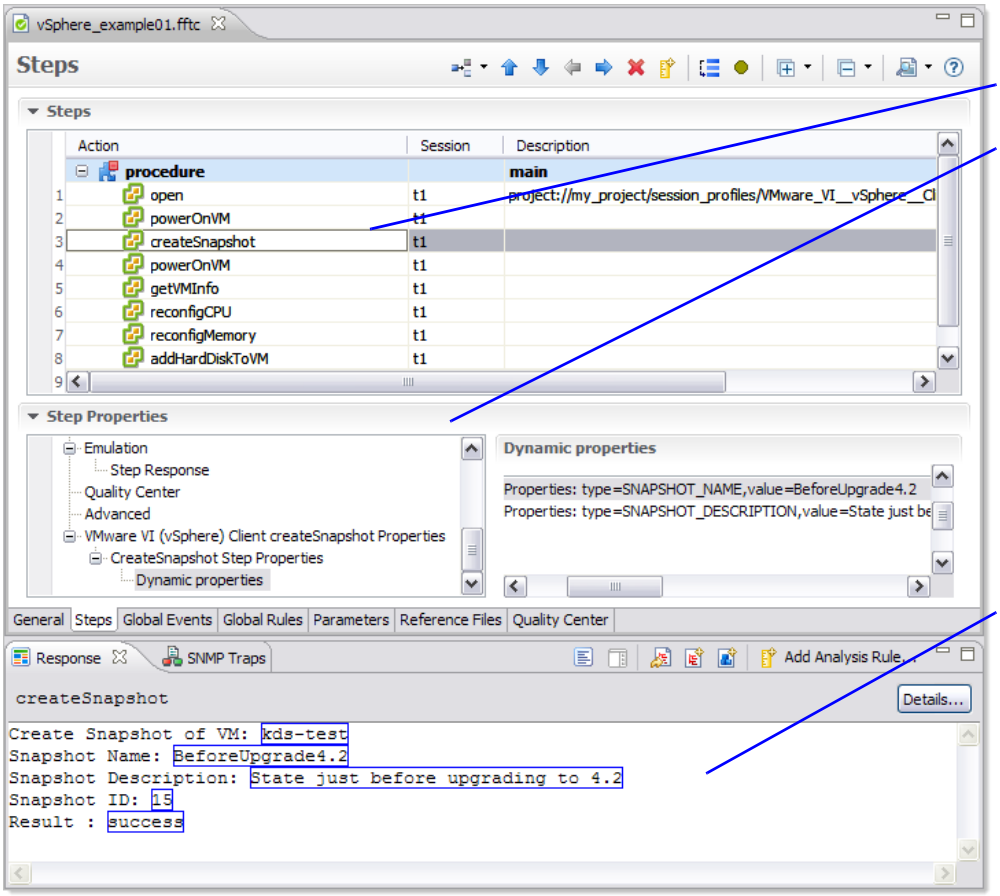
Notice that the **Result** text is "`success`". Remember that it would be easy to add an analysis rule to verify that the value of "`Result`" is "`success`" — iTest auto-generated a query that returns the value.

8 The snapshot is listed on the **Snapshots** tab. Select the snapshot to display its name and description.

4 Continue working in the interactive vSphere session. When you finish, close the session window and then click Finish Capture to save the captured steps into a test case. (The **Add Test**

Case wizard opens and help you through the process. For details on saving captured steps to test cases, see “Creating a test case by capturing interactive sessions” on page 174.)

5 The Test Case editor opens the new test case. Let’s look at the **createSnapshot** step in the test case and the response returned by the vSphere server.



We select the captured **createSnapshot** step and notice that the values that we typed for the name and description (the arguments to the action) are saved as property settings for the step.

You can parameterize most property settings so they can be set dynamically at runtime.

The response from the vSphere server is auto-mapped, so it is easy to use the auto-generated queries in an analysis rule to extract values and validate data items in the response.

6 You might now save this test case as the “setup the testbed” procedure.

You can then open additional iTest sessions of appropriate types (SNMP, Telnet, Web, and so on) with any of the VMs and proceed to test your application as needed. Once you finish, you can capture those steps as the procedure that validates the operation of your application.

Managing hosts, datacenters, datastores, resource pools, and VMs

For each task that you perform in an interactive session:

1 iTest submits the request to the vSphere server.

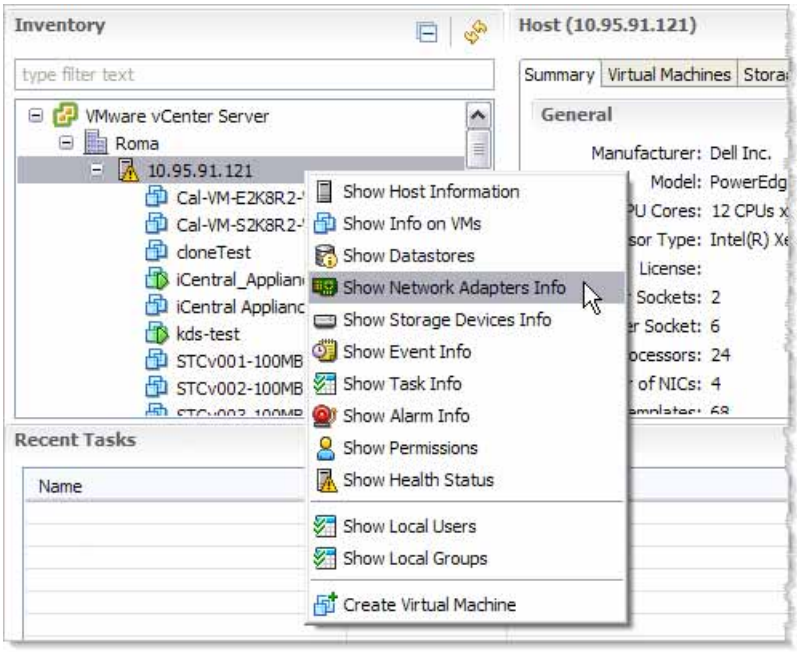
2 iTest captures the action (it might later become a step in a test case).

3 The server performs the task and returns a response.

4 iTest captures the response. Because you might later analyze the response to the step in a test case, iTest auto-generates queries and applies them to the response to return the data values of interest in the response—that is, iTest auto-maps all vSphere responses.

About tasks in an interactive session and the resulting iTest actions

- In a iTest session with vSphere, the processes of creating and managing elements follows the same flow as in vSphere. If iTest opens a dialog box to allow you to specify settings to perform the task, follow the procedure that you use in vSphere. For more details on any dialog box, see the VMware online help.
- Many actions for hosts, datastores, resource pools, and VMs appear on the context (right-click) menu for the selected item. In this example, we right-clicked a host and selected the **Get Network Adapters** action.



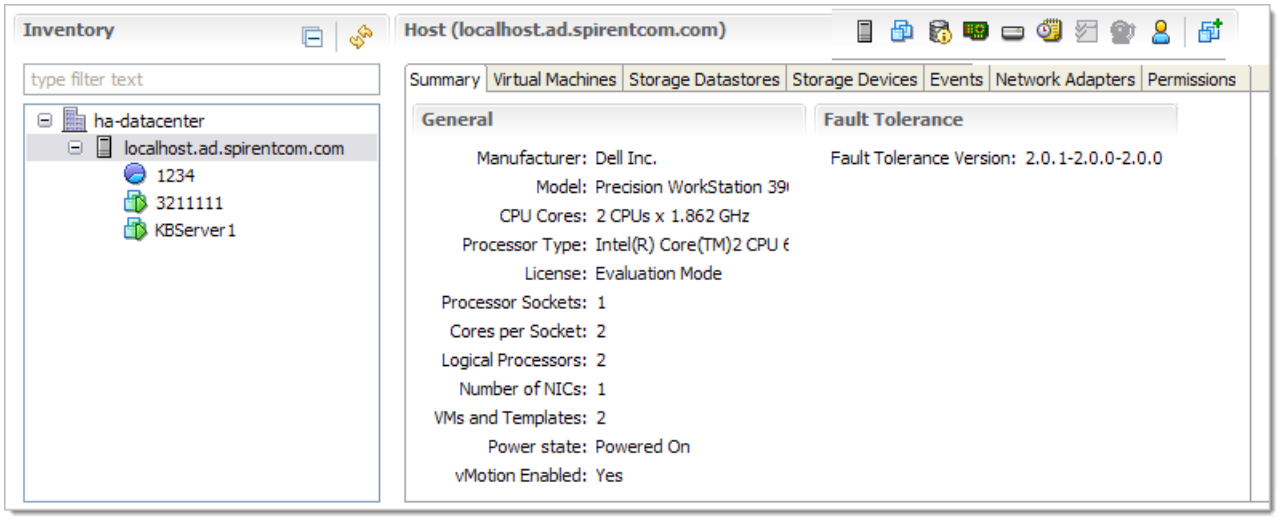
- For actions that involve a wizard or dialog box, the action is captured when you click **OK** or **Apply**, but not when you click **Cancel**.
- The item you select and the configuration setting values that you type (the arguments) for the command are saved as property settings for the resulting step in the iTest test case. You can parameterize most iTest property settings so they can be set dynamically at runtime.

  When you work with any step in the iTest Test Case editor that resulted from a captured action, open the **Step Properties** section to view the list of arguments for the action.

Datastores and hosts

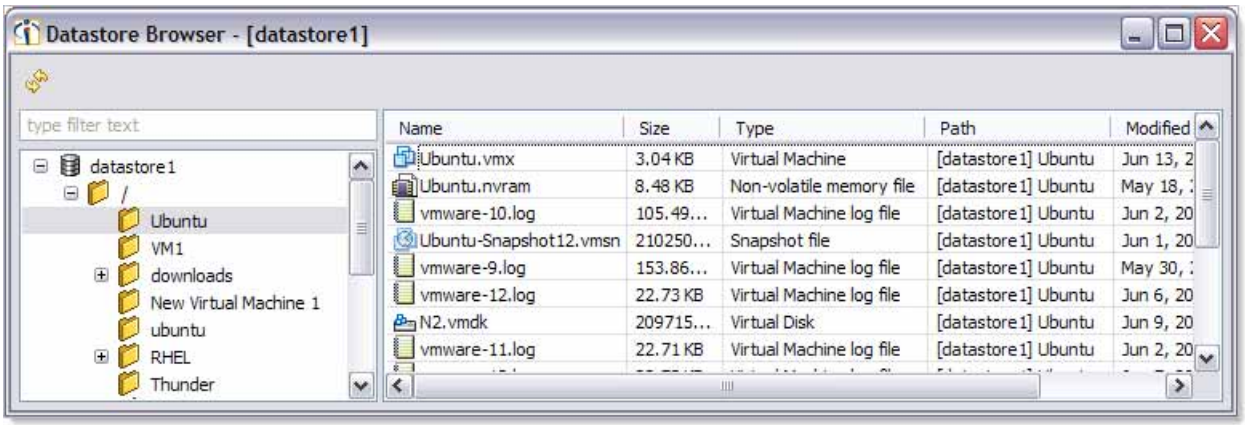
Select a datastore or host in the inventory tree. Each tab lists the configured items for the selected host.

Note The **Alarms** tab and the **Events** tab (and the associated "**Get**" buttons) appear when you select a host or datacenter, but only if you are logged in to a vCenter server.



- **To browse a datastore**

  Click the **Storage Datastores** tab to view the list of datastores. Right-click a datastore and select **Browse Datastore** to view the list of datastore contents (no action is captured).

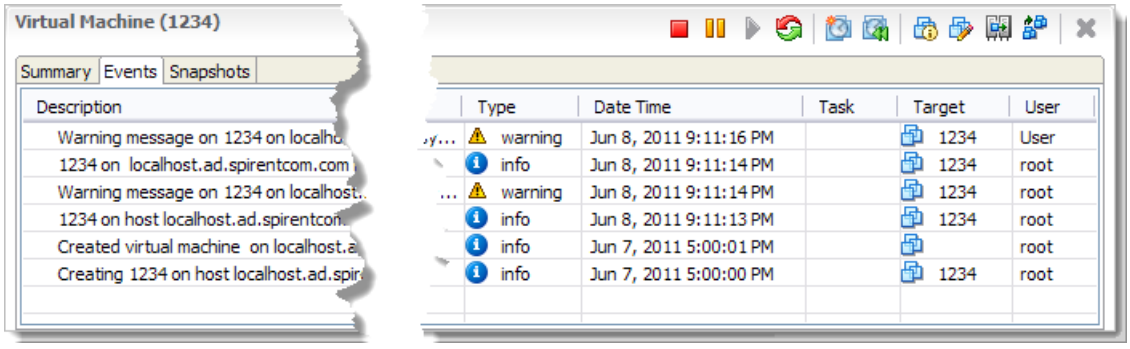


Events

Select an item in the inventory and then click the **Events** tab to view a listing of all tasks, events, warnings, and errors for the selected item.

- Double-click an event to submit and capture a **getEventInfo** action.

- To refresh the listing, right-click in the tab and select **Refresh Events** (no action is captured).



Snapshots

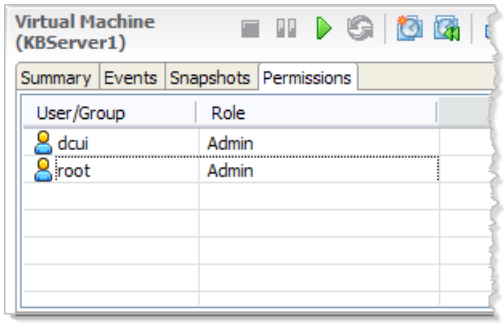
In an iTest session with vSphere, the processes of creating and managing snapshots follows the same flow as in vSphere.

Note To refresh the listing on the **Snapshots** tab: Select a VM, right-click in the **Snapshots** tab and select **Refresh Snapshots** (no action is captured).

Permissions

Select an item in the inventory and then click the **Permissions** tab to view a listing of each user and their permission setting.

To refresh the listing, right-click in the tab and select **Refresh Permissions** (no action is captured).



Tasks

The **Recent Tasks** pane displays the log of actions for the current vSphere session.

The **Tasks** tab appears only if you are logged in to a vCenter server.

Alarms

Note The **Alarms** tab and the **Events** tab (and the associated "**Get**" buttons) appear when you select a host or datacenter, but only if you are logged in to a vCenter server.

vSphere action reference

In a iTest session with vSphere, the processes of creating and managing elements follows the same flow as in vSphere. If iTest opens a dialog box to allow you to specify settings to perform the task, follow the procedure that you use in vSphere. For more details on any procedure, see the VMware online help.

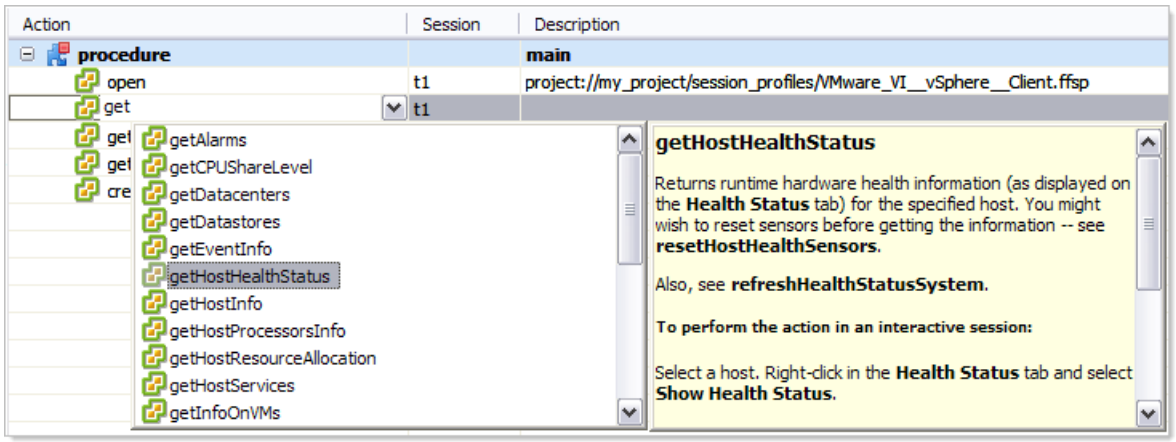
Typically, the actions for steps are captured during vSphere sessions, as described in “Creating a test case from an interactive vSphere session” on page 1641.

To add an action

Click in the **Action** cell for a vSphere step and then select the action from the drop-down list.

To view documentation for an action

Type the action’s name into the **Action** cell for a vSphere step — the description pops up when you select the action in the list.



To edit or view the arguments for the action (as properties of the step in the test case)

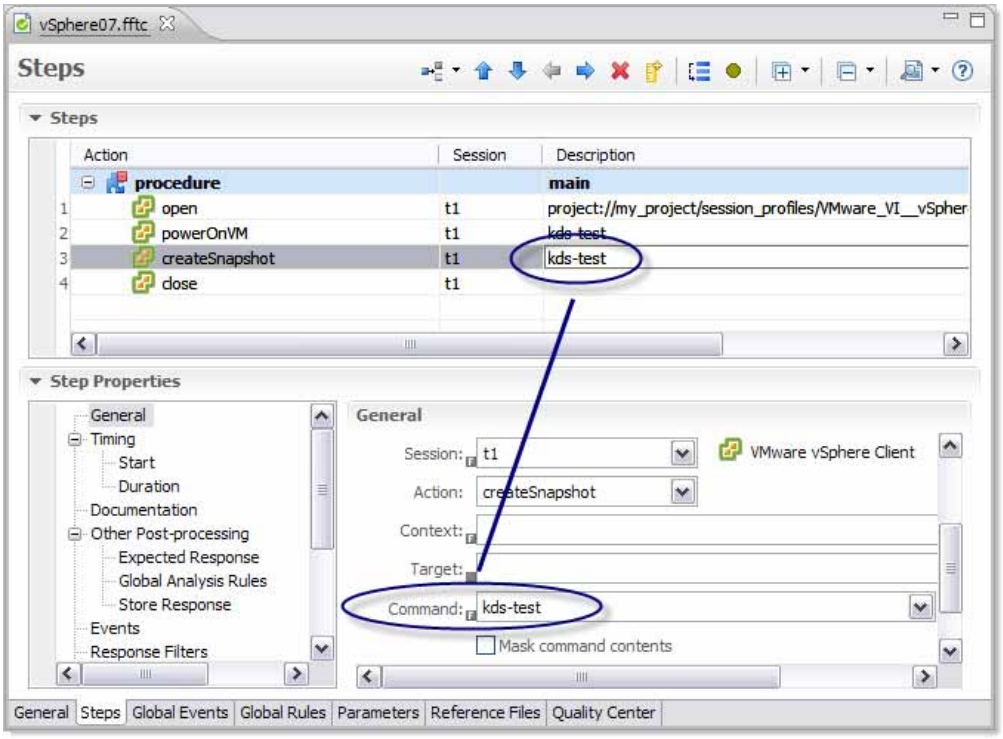
In a iTest session with vSphere, the processes of creating and managing elements follows the same flow as in vSphere. The arguments that iTest uses for an action are identical to the arguments used for the associated command in the vSphere API. For more details on any arg, see the VMware online help.

In a iTest test cases, you specify arguments for actions in the properties for the step. For example, the **createSnapshot** action has three arguments:

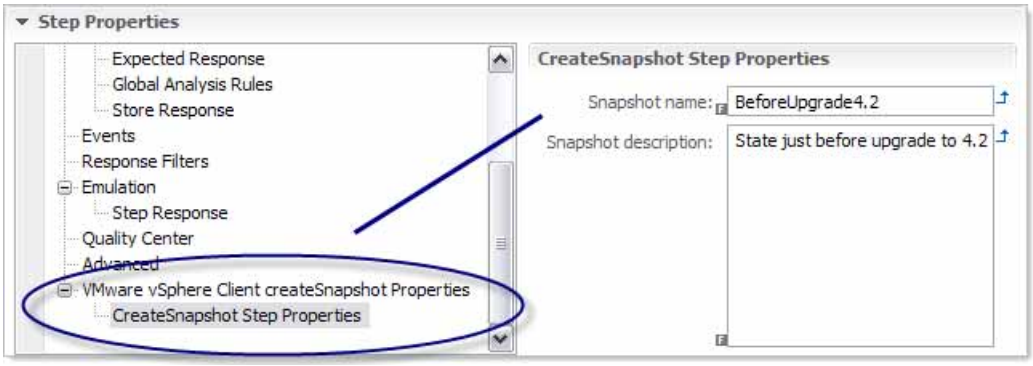
- The first argument is the VM name (the VM that you selected when performing the action during the interactive session).

  On the **Steps** page of the Test Case editor, select the **createSnapshot** step. Notice that the VM name appears in the **Description** cell for the step. For a vSphere step, the **Description** cell displays the value of the **Command** property where the first argument is specified. In

the **Step Properties** section, navigate to the **General** properties group to view or edit the setting for the **Command** property.



- The next two property settings are dynamic — they configure the snapshot: the **name** and **description** of the snapshot. In the **Step Properties** section, navigate to **VMware vSphere Client createSnapshot Properties > createSnapshot Step Properties**. View or edit the settings as needed.



Session profile property settings for VMware vSphere sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

vSphere session properties

vSphere server	Specify the hostname or IP address of the server.
Username	Specify the account name to use to log in to vSphere.
Password	Optional. Specify the password for the account.

CHAPTER 90

Selenium sessions

Overview: Selenium session window

When you start a Selenium session, iTest opens an instance of the Firefox browser. You interact with the pages in the normal way while iTest captures your actions and responses from the session.

The browser includes a context menu that enables you to perform any of several iTest operations (for example, capture the DOM of the HTML and a screenshot of the page or determine the name of a button or other element on the page).

Selenium sessions use the Firefox browser while you are performing a test manually — this is the session that iTest is capturing. You can specify a different **Replay Browser** (the browser to use when replaying the captured steps or when executing an automated test case). See also https://www.seleniumhq.org/ for details.

Limitations

- Selenium sessions do not support testing Flash applications that were developed using Adobe Flex.
- **clearCache** typically used to clear the cache from a previous test case execution, is not supported. In contrast, Selenium sessions open the browser for replay with a clean cache.
- For the **describe** and **getInnerText** actions, the element's inner text might differ from the equivalent item in iTest sessions.

In this chapter

To start a Selenium session

On the Session Profile editor (the **New Session** page):

1. Specify the URL of the page where the session should start. This is the information that you typically type into a browser's Address bar.

   To use a literal IPv6 address in a URL:

   - Disable field replacement (substitution) for the property.
   - As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

2. You have the option to specify the following property settings (there are several other settings — see "Session profiles: Session configuration settings" on page 120):

   - The **Replay Browser** (the browser to use when replaying the captured steps or when executing an automated test case)
   - The authentication credentials (**User ID** and **Password**) that iTest should submit to gain access to the site

3. Click **Save** to save the session profile and then click **Start**.

   iTest opens a session window and a Selenium IDE window in the background (you do not perform actions in either window). When the browser window opens, you can start performing the actions that you want to capture for the test.

How capture works in Selenium sessions

Actions

iTest captures the ***actions*** you perform in the browser (for example, when you type text into a text box, iTest captures a **type** action). When you save the captured step to a test case, the **Action** for the step is **type**.

Targets

Actions have ***targets***. A target is the control or element on the page that the action applies to (in our example, the name of the text box to type into: **name=address1**). The targets on a Selenium page are stored in a ***form map***. While developing a test case, you will make use of the information in the map to perform an action on a target — in our example, you perform the **type** action on the text box that is named "**name=address1**". Another common example is to perform a **click** action on a button (the name or identifier of the button is the target).

More detail in "Selenium Targets and Commands" on page 1675 and 7"Overview: Form Maps" on page 843.

Commands

Some Selenium actions include arguments that are captured as iTest ***commands***. In our example, typing the text "**123 Main Street**" into the text box and then clicking in another part of the page results in a **type** Action with the Target **name=address1** and a Command of **123 Main Street**

About ‘prompts’ in Selenium sessions

For Selenium sessions, a *prompt* is the text that appears in the body of a prompt dialog box. For example, an info box might pop up to tell you to set a particular value before proceeding. iTest caches (saves) the text that appears in the box (that is the **prompt** value).

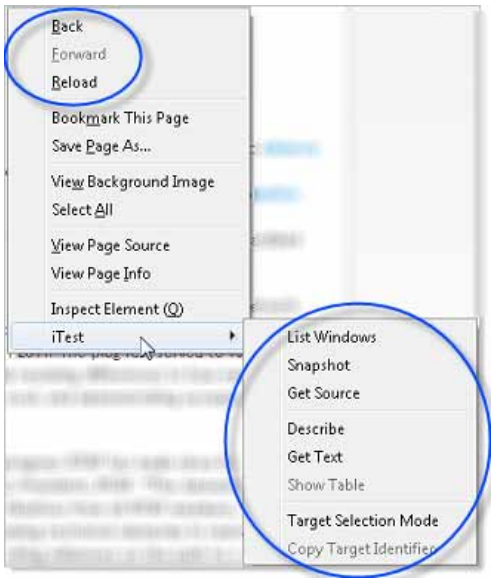
iTest caches the text of each prompt box that appears during test case execution. The prompt box typically includes an **OK** or **Cancel** button that you click after reading the **prompt** text. There are three types of prompt dialog box:

- **Alert**: Makes you aware of a particular condition. You read the **prompt** text and click a button to proceed
- **Confirmation**: Confirms an action that you just took. You read the **prompt** text and click a button to proceed
- **Answer**: You read the **prompt** text (iTest caches (saves) the text of each prompt that appears during test case execution), type text into a text box, and then click a button to proceed. iTest captures any text that you type and saves it as the text of the **Command** property for the resulting step.

Using the context menu to capture actions on elements

You can capture many Selenium actions by right-clicking an element and selecting the action in the context menu. Most of the actions return data for analysis in the test case. This is particularly helpful when working in popups or Selenium prompts.

(If the application uses right-click for another operation, use Shift+right-click.) The circled actions are captured:



Back / Forward / Reload	Go back to previous page Go forward to next page Reload the current page
iTest > List Windows	Lists all active windows, both main window and popup windows that Selenium steps launched. Identifies the currently selected window. Typically used to return data for analysis. Displays the frame structure for each window, including nested windows. For every frame in every window, obtains the following values from the browser's page information and adds the values to the structured data for the response. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze values of interest in the response. • Title of the page • URL • Window name • Frame name • Content type (value defaults to empty) • Encoding • Referring URL • Last modification date • Metadata for tags
iTest > Snapshot	A "snapshot" in Selenium sessions is made up of both a visual image of the page and the DOM of the page. Typically used to return data for analysis. The **snapshot** action performs the following actions: • Return the XML representation of the HTML on the page (the DOM) as the response to the **snapshot** action. This is useful for reviewing the HTML code for the page. A snapshot action also enables you to create a Form map for the page. See "Overview: Form Maps" on page 843. • Saves a graphic image of the page. The image appears in the test report
iTest > Get Source	Returns the HTML source of the current HTML page. This is the equivalent of clicking **View > Source** in a browser. If you right-click a frame in the Target, then **Get Source** returns the source of the document in the frame.
iTest > Describe	Returns a list of attributes for the specified target, organized into a table. For example, you can use **Describe** to: • Determine the size or content of text boxes • Determine whether radio buttons or check boxes are selected.
iTest > Get Text	Extracts the text from within the HTML structure for the specified node (target).
iTest > MouseOver	Captures the **mouseOver** action. Selenium sessions do not auto-capture all mouseOver events (they might unnecessarily clutter a test case). Use this feature to explicitly capture a **mouseOver** action.

iTest > Show Table	For the selected table, returns all table contents in the form of a table. Only text is returned, images and merged cells in the source are ignored. When no arguments are specified, displays the whole table. If the specified number of rows and columns exceeds the limits of the data, then, the whole table is displayed. Indexes for row and column start at 1.
iTest > Target Selection Mode	When you select this option, iTest enters a mode that identifies targets on the page. Target Selection mode is useful when you need a Target ID (for example, to paste it into a test case step) but do not want to spend the time to capture a session. While in Target Selection mode: • Hover the cursor over a target to highlight it (there may be a moment's delay) and to view its Target ID in the status bar of the Web browser • Double-click a target to add its Target ID to the clipboard. This option is useful while you are working on a test case step. First, use this option and then paste the Target **ID** value directly into the Test Case editor. If double-click does not work, then use **Copy Target ID**. To exit Target Selection Mode, open the context menu again and uncheck **Target Selection Mode**.
iTest > Copy Target ID	Right-click or Ctrl+left-click an element (node) to open the context menu and select **Copy Target ID** to copy its Target ID to the clipboard. This option is useful while you are working on a test case step. First, use this option and then paste the Target **ID** value directly into the Test Case editor.

Creating Selenium test cases

As with most session types, the easiest way to create a Selenium test case is to manually execute the steps that you expect to include in the test case and then save the captured session into a test case.

When you start a Selenium session, iTest opens an instance of the Chrome or Firefox browser. For example, the Selenium IDE for Chrome or Selenium IDE for Firefox in the test case step. You interact with the browser in the normal way while iTest captures your actions and the responses from the session.

Tip Use the iTest context menu to capture a variety of actions. See "Using the context menu to capture actions on elements" on page 1654.

1. Manually execute the steps that you expect to include in the test case.
2. Take a snapshot of each web page that you expect the test case to interact with (that is, a step will click a link or button, type text into a text box, and so on). This enables you to generate form maps for the pages.
3. Create a form map library and save all of the form maps into it. Form map libraries are described in more detail in "Form map libraries" on page 858.
4. Edit the session profile or device for the session so that you associate the form map library with the session profile or device. Save the updated session profile. (Details appear in "Associating a form map library with a session" on page 858.)

5 Save the captured session as a procedure in a test case.

Selenium steps differ from steps for other session types. For most other session types, The **Description** cell on the **Steps** page displays the **Command** property value. In some Selenium steps, the **Description** cell displays the following values, separated by colon and semicolon:

```
Context:Target; Command
```

6 Edit the steps as needed. For example, replace Target queries with easier-to-remember Target names, and so on.

Handling unexpected popup windows

1 Design the test case as if no popups will appear. Popups appear, for example, to report an event or a change in expected status such as an error.

2 Add a "popup detection" step after each step that might result in a popup. Use the **listWindows** action to test for a popup. **listWindows** returns the count of windows. Example response:

```
Current Window: MainWindow
There are 2 windows open:
Window 1: configWin
Window 2: MainWindow
...
```

3 Add an analysis rule that determines whether there are too many windows showing. Because **listWindows** returns a list of window names, you can identify the popup.

About mixing browsers

You must use Firefox to capture a session. You can execute the resulting iTest test case in either Firefox or Internet Explorer. See the description of the **Replay browser** session property in "Replay > Browser" on page 1671.

Executing Selenium sessions

During automated execution, the browser (Chrome or Firefox) status bar indicates Selenium execution by displaying the words "Web Driver".

Selenium Action reference

All of the actions listed in the "Selenium action reference" table on page 1659 are available in Selenium session steps in test cases. Each entry in the table describes the function of the action and specifies whether settings are required for the **Target** and **Command** properties for the step. See also https://www.seleniumhq.org/ for details.

Capturing Selenium actions

Not all actions are "capturable" or executable — see "Selenium actions that iTest can capture and execute in test cases" on page 1674.

Performing Selenium actions during interactive sessions

During interactive sessions, you can perform many Selenium actions using the context menu. See "Using the context menu to capture actions on elements" on page 1654

Step properties associated with all Selenium actions

All Selenium actions have the step properties that are listed in this section. For some actions, additional properties are described in the "Selenium action reference" table on page 1659. (In the **Step Properties** section of the Test Case editor's **Step** page, navigate to the **Selenium <actionName> Properties** group. For example, for the **click** action: **Selenium click Properties**.)

HTTP credentials

- **User:** Optional. Specify the user account name used to connect to the Selenium site
- **Password:** Optional. Specify the password used to connect to the Selenium site. The text is masked here and in all locations in which it is used.

Note Credentials that you specify here are applied only for **open** and **go** steps.

Wait time

Maximum time to wait for target: Specify the maximum number of seconds to wait for the target to appear before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing.

Default: 15

Maximum time to wait for popup: Specify the maximum number of seconds to wait for the popup to appear before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing.

Default: 15

Completion

Maximum time to wait for a page load: Specify the maximum number of seconds to wait for the requested page to load before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing.

Default: 120

Selenium action reference

Action Name	Target	Command	Notes
addSelection	Required	Required	**addSelection** is similar to **select**, and is used for multi-selection lists. **addSelection** adds to the current selection, but does not unselect the existing selection (**select** steps do unselect the existing selection). The **Command** property specifies the set of labels to select in addition to the current selection. Use commas to separate labels. **Note** This action is an alias for the **selectAdditional** command in iTest Web sessions
answerOnNextPrompt	Required string to send in answer	Not Required	Sends the **answerString** specified in the **Target** property as the response to the next Answer type of prompt. The prompt may appear after several subsequent action steps, "next" means that it is the first after the execution of step with this action — only one next prompt is answered. **Note** This action is a replacement for the **answerPrompt** command in iTest Web sessions. There is one important difference: The answer text for the **answerOnNextPrompt** action is supplied in the **Target** property. In contrast, the answer text for the **answerPrompt** action appears in the **Command** property. For Selenium sessions, a prompt is the text that appears in the body of a prompt dialog box. See "About 'prompts' in Selenium sessions" on page 1654.
answerPrompt	Not Required	Required string to send in answer	**answerPrompt** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. **Note** The Selenium **answerOnNextPrompt** action is a replacement for the Web **answerPrompt** command. There is one important difference: The answer text for the **answerOnNextPrompt** action is specified by the **Target** property. In contrast, the answer text for the **answerPrompt** action appears in the **Command** property.
chooseCancelOnNextConfirmation	Not Required	Not Required	When the next confirmation dialog appears (dialog may appear after several further action steps, "next" means that it is the first after execution of step with this action), **chooseCancelOnNextConfirmation** clicks **Cancel** (only one next dialog is answered) **Note** The default action for Selenium sessions is **chooseOkOnNextConfirmation** For Selenium sessions, a prompt is the text that appears in the body of a prompt dialog box. See "About 'prompts' in Selenium sessions" on page 1654.

Action Name	Target	Command	Notes
chooseOkOnNext Confirmation	Not Required	Not Required	When the next confirmation prompt dialog box appears (dialog may appear after several further action steps, "next" means that it is the first after execution of step with this action), **chooseOkOnNextConfirmation** clicks **OK** (only one next dialog is answered) **Note** The default action for Selenium sessions is **chooseOkOnNextConfirmation** For Selenium sessions, a prompt is the text that appears in the body of a prompt dialog box. See "About 'prompts' in Selenium sessions" on page 1654.
clearCache	—	—	**clearCache**, a rarely used action in iTest Web sessions, was typically used to clear the cache from a previous test case execution. In contrast, Selenium sessions open the browser for replay with a clean cache. **clearCache** is not supported in Internet Explorer 10.
clearPrompts	Not Required	Not Required	See "About 'prompts' in Selenium sessions" on page 1654. The **clearPrompts** command removes all prompts from the cache. See the **waitForPrompts** command.
clearText	Required [input field]	Not Required	Sets the value of an input field to <empty>. See **setText**.
click	Required	Not Required	Clicks an element (HTML object) like a button, link, radio button, or check box on the current page. **Note** Do not use **click** for combo boxes, instead, use **select** or **selectAdditional**. **Keys**: Check a box to include the specified key with the click (for example, **Shift+click**). **Coordinates:** Specify the X and Y coordinates in pixels. Positive X is to the right and positive Y is down. **Client coordinates**: The origin is the upper left corner of the bounding rectangle of the element. **Screen coordinates**: Not implemented.
closeWindow	Not Required	Optional	Closes the specified window. If no window is specified, closes the current window. The **Target** property specifies the window to close. If **Target** is empty, then the currently selected window is closed.
confirm	—	—	**confirm** is deprecated in Selenium sessions and is not captured. The **confirm** action exists only for backward compatibility with existing Web test cases.
credentials	—	—	This special command is inserted when HTTP credentials are submitted while capturing a manual session. When iTest renders captured actions into a test case, the **credentials** action is removed and the captured credentials are added into the step properties of the preceding step. Do not use the **credentials** action in test case steps.

Action Name	Target	Command	Notes
describe	Required	Not Required	Returns a list of attributes for the specified target, organized into a table. Use **describe** to: Determine the size or content of text boxes Determine whether radio buttons or check boxes are selected.
dragAndDrop	Required	Not Required	Drags a source web element and drops it into the target. The **Context** property specifies the drag element. The **Target** property specifies the target (the drop element). **Tip** If the drag and drop element loads slowly, add a **sleep** step before the **dragAndDrop** step.
eval	Not Required	Required	Executes the specified JavaScript in the context of the selected window. The **Command** property specifies the JavaScript code to evaluate.
fireEvent	Required	Not Required	For the specified element, simulates the specified JavaScript event. (Specify the event name to trigger the corresponding "onevent" handler.) For **fireEvent** steps, we recommend that you specify a non-zero Idle time. This gives the listeners time to handle the event appropriately. **Send mouse events**: Some applications require three events for a click: **mousedown** followed by **mouseup** and then **mouseclick**. **Keys**: Check a box to add the specified key with the click (for example, **Shift+click**). **Coordinates:** Specify the X and Y coordinates in pixels. Positive X is to the right and positive Y is down. **Client coordinates**: The origin is the upper left corner of the bounding rectangle of the element. **Screen coordinates**: The origin is the upper left corner of the screen.
formSubmit	—	—	**formSubmit** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **submit** action is an alias for **formSubmit**. See **submit**
getInnerText	Required	Not Required	Extracts the text from within the HTML structure for the specified node (Target). **<nbsp>** tags appear as whitespace iTest adds newlines to improve the appearance of the response. iTest adds newlines for ** ** and any tag that causes newlines.

Action Name	Target	Command	Notes
getSource	Not Required	Not Required	Returns the HTML source of the current HTML page. This is the equivalent of clicking **View > Source** in the browser. If you specify a frame in the **Target** property, then **getSource** returns the source of the document in the frame. **Note** The response is text only and differs for different browsers. Do not use the **getSource** action when verifying DOM structure -- instead, use the **snapshot** action. **Examples** ActionTarget Command getSource <None> <None> getSource frame3 <None> **Note** The **getSource** action is an alias for the **showPage** command in iTest Web sessions.
go	Not Required	Required	Navigates to the URL specified in the **Command** property. During an interactive session, if you type a new address in the address bar and press **Enter**, it is captured as a **go** action.
goBack	Not Required	Not Required	Simulates the user clicking the **Back** button on the browser.
goForward	Not Required	Not Required	Simulates the user clicking the **Forward** button on the browser.
listPrompts	Not Required	Not Required	See "About 'prompts' in Selenium sessions" on page 1654. A **listPrompts** step returns the text of all prompts that have appeared since the Selenium session started or since the last **clearPrompts** command. The response is structured and iTest auto-generates queries that enable you to extract the text. To enable **listPrompts** on Internet Explorer versions 8, 9, or 10: 1. Click **Tools > Internet Options** Perform the following steps for all zones: 2. On the **Security** tab, click **Custom Level** 3. Under the **Miscellaneous** group, set **Display mixed content** to **Enable**
listWindows	Not Required	Not Required	Lists all active windows--main window and popup windows--that Selenium session steps launched. Identifies the currently selected window. Displays the frame structure for each window, including nested windows.
mouseOver	Required	Not Required	A **mouseOver** step simulates a mouse-over action on the specified **Target**. If the target has mouse-over handlers, the handlers are invoked.
refresh	—	—	**refresh** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **reload** action is an alias for **refresh**. See **reload**

Action Name	Target	Command	Notes
reload	Not Required	Not Required	Reloads (refreshes) the page on the browser. **Note** This action is an alias for the **refresh** command in iTest Web sessions
removeSelection	Required Name of item to unselect	Required	Un-select specified list items. You can specify the item by name, by alias, or by index. Index is 0-based. Use commas to separate items.The **Command** property specifies the set of labels to un-select. **Example** Action Target Command removeSelection ListOfOptions index=2,4,7,9 removeSelection ListOfOptionsindex=4:9 (4:9 or 4,9 represent indexes 4 though 9, inclusive) **Special cases** To select content that includes the equal, colon, or comma characters (= : ,), use the backslash character \ to escape the character. For example, to un-select "23:18", use "23\:18". ? marks a single character in the content as wild. * marks the remainder of the string as wild. **Note** The **removeSelection** action is an alias for the **unselect** command in iTest Web sessions
runScript	Required	Not used	Executes the specified JavaScript in the context of the selected window. The **Target** property specifies the JavaScript code to evaluate. Example of action **runScript** command: `window.foo = function foo() {document.body.innerHTML="hello";} ; window.foo()` **Note** This action is an alias for the **eval** command in iTest Web sessions
select	Required	Required	Use **select** to select items in a list or a combo box. The **Target** is the combo box or list box in which you make the selection. The **Command** property specifies the labels to select in the box (for example, label=myLabel). Use commas to separate labels. **select** steps also unselect anything that had previously been selected. **Note** Because items are captured by label, you normally do not need to select by index. If you have an item's index and need its label in order to select it, use snapshot. In the snapshot structured data, run the query "{xpath to the select combo}//option" This will give you all possible items. "{xpath to the select combo}//option[index]" should give you the label for the particular index.

Action Name	Target	Command	Notes
selectAdditional	—	—	**selectAdditional** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **addSelection** action is an alias for **selectAdditional**. See **addSelection**
selectWindow	Not Required	Required	**selectWindow** changes the focus to the specified popup window. Steps can then perform additional actions on the window. The **Target** property specifies the name of the window to select. To go back to the main window, use **selectWindow** with a **Target** property value of **MainWindow**.

setFilePath	—	—	**setFilePath** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases.
			Use the **type** action with the **file upload** field set as a Target to pick the file by the specified path. The **type** action is the only action captured when picking a file while capturing a session.
			Notes:
			• Only one **setFilePath** in the script is allowed
			• **setFilePath** must contain file download location (with the file name)
			  • **Windows**: Use " \ \ " as the separator for the download path
			  • **Linux**: Use / as the separator for the download path
			• A manual wait for the download to finish is required (for example using the **sleep** command)
			Internet Explorer / Chrome
			• The target is taken from the next **click** step. The download link is extracted heuristically from the matching element. Alternatively, you can specify th e download link in the setFilePath Target field. It must be an URL, not a locator.
			• You must use setFilePath just before the **click** step that is causing the download unless you specify an explicit download URL in the setFilePath.
			• The click step that follows the setFilePath step is not included in the test report
			Firefox
			• The file name in the download location is ignored. Firefox assigns a name to the downloaded file.
			• Only specific mime types can be downloaded:
			  • "application/pdf" and similar for the pdf
			  • "application/" for the *.cfg from the Netgear testcript

setText	—	—	**setText** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **type** action is an alias for **setText**. See **type**

Action Name	Target	Command	Notes
showPage	—	—	**showPage** is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **getSource** action is an alias for **showPage**. See **getSource**
showTable	Required tableLocator	Optional	For the specified table, return all table contents in table form. Only text is returned, images and merged cells in the source are ignored. Columns are delimited using the tab character. **Step Properties** **Command**: Specifies the ranges of rows and columns to return. Use the following format. startingRow startingColumn numberOfRows numberOfColumns Index numbers are 1-based (the first row is number 1, and so on). If **Command** is empty or if the specified number of rows and columns exceeds the limits of the data, then the entire table is returned. **Example** ActionTarget Command showTable portTable1,1,6,3 **Include child nodes**: Check the box to include the child nodes of table cells in the structured data in addition to the text of the cells. Default: unchecked (do not include child nodes) **Example** For the following table: <table><tr><td>HelloWorld</td></tr></table> **Include child nodes** unchecked returns: <table><row><cell>Hello World</cell></row></table> **Include child nodes** checked returns: <table><row><cell>HelloWorld</cell></row></table>
snapshot	Not Required	Not Required	You typically use **snapshot** during interactive Selenium sessions to capture information about the targets on the page so that the automated test can take actions on the targets. A “snapshot” in Selenium sessions is made up of both a visual image of the page and the DOM of the page. The **snapshot** action performs the following actions: • Return the XML representation of the HTML on the page (the DOM) as the response to the **snapshot** action. This is useful for reviewing the HTML code for the page and for identifying Targets on the page. A snapshot action also enables you to create a Form map for the page. See “Overview: Form Maps” on page 843. • Saves a graphical image of the page. The image appears in the test report and you can display the image in the Images view.

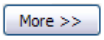
Action Name	Target	Command	Notes
submit	Required	Not Required	Rarely, you must submit a form that does not have a **Submit** button. You submit the form by pressing the **Enter** key in some field in the form. Use **submit** for such cases. The **Target** is the FORM element to submit. **Note** This action is an alias for the **formSubmit** command in iTest Web sessions
type	Required	Required	Sets the value of an input field as though you typed it in. If there is currently text in the field, then **type** replaces it. The **Command** property (**Description** cell) in the Test Case editor specifies the text to set in the text box in the following format: Context:Target; Command. If **Command** is empty, then an empty string is set. In Selenium sessions, use the **type** action with the **file upload** field set as a Target to set the file by the specified path. The **type** action is the only action captured when picking a file while capturing a session. **Notes** • In the form map, multi-line text boxes are under the <textarea> node. • This action is an alias for the **setText** command in iTest Web sessions
unselect	—	—	The **unselect** action is deprecated in Selenium sessions and is not captured. The action exists only for backward compatibility with existing Web test cases. The **removeSelection** action is an alias for **unselect**. See **removeSelection**
waitForPopUp	Required	Required	Waits until a popup window appears and loads (typically does something with the popup or with its contents). When a popup appears, then execution immediately proceeds with the next step. If a popup does not appear, then an **onWebStepSyntaxError** event is thrown. During capture, **waitForPopUp** is placed before actions inside a popup. In test cases, the Target is the popup name and the Command is the time in milliseconds to wait until the popup appears.

Action Name	Target	Command	Notes
waitForPrompt	Not Required	Not Required	See "About 'prompts' in Selenium sessions" on page 1654. Waits for the appearance of a popup prompt dialog box. When a prompt appears, then execution immediately proceeds with the next step. The response is the most recently cached prompt text. There are two possible conditions: • An immediately preceding **clearPrompts** command removed all prompts from the cache so that the **waitForPrompt** command will wait until a new prompt dialog box appears. • A prompt exists in the cache, so the **waitForPrompt** command immediately proceeds with the next step. Specify properties in the **Step Properties > Selenium waitForPrompt Properties** page for the step. See the **clearPrompts** command. Step properties: **timeout**: Specify the maximum time in seconds to wait for the message prompt. Default: 15.
waitForTarget	Required	Not Required	This action is useful for Ajax applications with dynamically loaded controls. For example, a combo box appears only after you click a button, and you don't know how long it could take for the box to appear. In this case, create a Selenium **waitForTarget** step and specify the combo box in the **Target** property. Specify the time to wait in the **Step Properties > Selenium waitForTarget Properties** page for the step.
windowMaximize	Not Required	Not Required	Maximizes the current window if it is not already maximized.

Session profile property settings for Selenium sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 120. For a detailed description of how iTest uses the settings, see "About property settings" on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click 

.

Selenium

URL of starting page	Specify the URL of the Selenium page at which to begin. This is the information that you typically type into a browser's Address bar. If you do not specify a URL, then the browser opens without going to a page. You can type the URL into the address box. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

Firefox Profile

If the iTest host has multiple profiles for a version of Firefox setup, use the **Firefox profile option** to specify the profile path to use in the Selenium sessions.

Firefox profile is the collection of settings, customization, add-ons and other personalization settings that can be setup on the Firefox Browser. You may customize Firefox profile to suit your Selenium automation requirement. To create or remove FireFox profile, see https://support.mozilla.org/en-US/kb/profile-manager-create-and-remove-firefox-profiles

Firefox profile path	Enter the Profilepath location. For example: • **For windows 7 >** `/AppData/MozillaFirefoxProfile_name.default` • **For Linux >** `/.mozilla/firefox/profile_name.default/` **Note** Firefox profile includes personal information such as bookmarks, passwords, and user preferences which can be edited, deleted or created using the program manager.
Import certificates	Select to import certificate from the selected FireFox profile. Imports all your certificate in that profile and then uses that profile while instantiating Firefox webdriver instance.

Snapshot

Maximum width to capture	If you select the Take a snapshot when a page loads property (in the Capture > Page Loads group), then you can use this property to set the maximum width of the digital image of the page. Default: 1024 pixels
Maximum height to capture	If you select the Take a snapshot when a page loads property (in the Capture > Page Loads group), then you can use this property to set the maximum height of the digital image of the page. Default: 800 pixels
Capture an image of the Selenium page when performing any action on the page	Capture a graphical image with any captured action. The image appears in the test report. Default: checked
Skip DOM Capturing	Select to skip DOM ("iTest > Snapshot" on page 1655) capturing, that is, disable HTML processing for **snapshot** command (both text and structured response will be empty). This option works for both capture and replay sessions. Select this option to optimize the memory usage for **snapshot** command, when you need only the screenshot.
PNG screenshot export path	Enter the path and file name to save the processed PNG screenshot from the current step to location specified. The following rules apply: • Parent folder must exist to save the snapshot. • File name to save the snapshot must be entered. • File extension: **.png**, must be entered to save the snapshot. If the above rules are not followed, the following validation messages display: • [INFO] File doesn't have .***png** extension • [WARNING] Parent directory doesn't exist • [ERROR] No file name specified **Note** If the folder does not exist, it is not created and screenshot will not be saved.

Snapshot>HTTP credential

User / Password	If the session will access a secure site, then provide the credentials that iTest should submit to gain access. • Firefox: If you set the properties here, in the session profile, then before starting the session, the browser will not prompt you to enter the credentials. If you do not set the properties here, then the browser will prompt you for credentials. • Chrome: If you set the properties here, in the session profile, then before starting the session, the browser will not prompt you to enter the credentials. If you do not set the properties here, then the browser will prompt you for credentials. • Internet Explorer: You must specify HTTP credentials here, in the session profile. **Note** These values are HTTP credentials. The values are not used when an application login page asks for credentials.

Snapshot>Wait time

Maximum time to wait for target:	Specify the maximum number of seconds to wait for the target to appear before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing. Default: 15
Maximum time to wait for popup:	Specify the maximum number of seconds to wait for the popup to appear before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing. Default: 15

List Prompts

Character set for URL encoded content	Select the character set from teh dropdown list for URL encoded content. Default: UTF_8

HTTP Credentials: See "HTTP credentials" on page 1658

Wait time: See "Wait time" on page 1658

Capture > Browser

If the iTest host has multiple versions of Firefox installed, use the **Capture > Browser > Browser path** property to specify the version to use while you capture interactive Selenium sessions.

Note You have the option to use the same or a different browser when replaying the captured steps or when executing an automated test case. See the **Replay > Browser > Replay Browser** property.

Browser Type	Specify the browser to use **Forefox** or **Chrome**, for manual capture and for automated execution of Selenium sessions. Default: Firefox
Browser path	Specify the browser to use for manual (interactive) Selenium sessions. Default: iTest finds and uses the most up-to-date version of browser on the iTest host.
Proxy	Select option to set HTTP proxy or FTP proxy based on your requirement. Options: Default, No, Automatic, System Default: Default

Replay > Browser

You can specify the browser that iTest should use when replaying captured steps or when executing an automated test case.

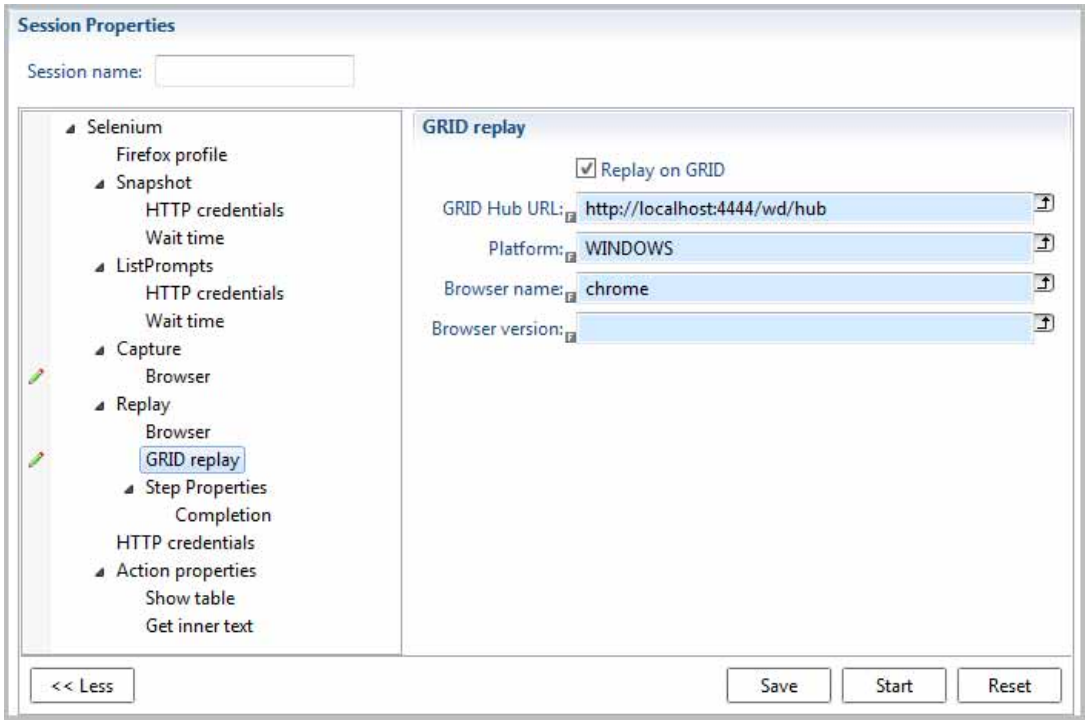
Note You also have the option to specify the version of Firefox to use while you capture interactive Selenium sessions. See the **Capture > Browser > Replay Browser** property.

Replay browser	Specify the browser to use for manual replay and for automated execution of Selenium sessions. Default: Firefox
Browser path	If the iTest host has multiple versions of the browser installed, then specify the path of the version to use. **Note** Ignored when Internet Explorer is specified.
Download path	Specify the default directory/path to download the file for replay (captured file).

Replay > Grid

The Selenium Grid extends the Selenium session replay properties, where you can override the test case to replay on Grid (regardless of whether the Selenium server is internal or external to the iTest server). The Grid replay provides you with an ability to run Selenium test cases on a Selenium Grid server by setting various options like GRID Hub server URL, Platform OS, Browser Name and Version from the session profile or the topology.

Note The current iTest release support Selenium Grid version 2.44.0 (see also Selenium Grid document: https://code.google.com/p/selenium/wiki/Grid2).



Replay on GRID	Select to override the test case to replay on the Grid **Note** When GRID replay is enabled, iTest inits remote webdriver instance (which points to the provided GRID Hub and sends the platform/browser settings), instead of local settings.
GRID Hub URL	Specify the GRID Hub URL
Platform	Specify the OS to which the replay is restricted.
Browser name	Specify the browser to which the replay is restricted. The default valid Grid browsers are: **Chrome**, **Firefox** and **Internet Explorer**
Version	Specify the browser version to which the replay is restricted.

Limitations

Supported	Not Supported
Only one version of Selenium GRID Hub in the current release (however, a broader range of versions may work fine).	**clearCache** command (does nothing during Grid playback for any browser).
Only the browsers/platforms supported by the local Selenium session are officially supported for GRID replay.	HTTP/HTTPS authentication
	Popup logging
	For Firefox, no custom profile is created on test case replay startup (so, no support for the following): • specifying download dir • unstable page loading • overriding MIME types • custom certificates

Note All the regular Selenium local replay limitations also apply to GRID replay.

Replay > Step Properties > Completion

Maximum time to wait for a page load	During replay: For some steps to complete execution, the page that the step attempted to load must be loaded. Specify the maximum number of seconds to wait for the requested page to load before performing an action on it. If the time is exceeded, iTest declares an execution issue for the step and then continues executing. Default: 120 seconds
Do not wait for the page to be loaded completely	Select to improve performance. **Note** Selecting this option may lead to stability issues. Default: not checked

HTTP Credentials

User / Password	If the session will access a secure site, then provide the credentials that iTest should submit to gain access. • Firefox: If you set the properties here, in the session profile, then before starting the session, the browser will not prompt you to enter the credentials. If you do not set the properties here, then the browser will prompt you for credentials. • Internet Explorer: You must specify HTTP credentials here, in the session profile. **Note** These values are HTTP credentials. The values are not used when an application login page asks for credentials.

Selenium actions that iTest can capture and execute in test cases

iTest captures the following types of Action in Selenium sessions (see "Selenium Action reference" on page 1657 for full descriptions):

Tip During capture, if you encounter a "Log in" page and use multiple incorrect passwords, then, to save time during replay of the captured session, iTest executes only the final correct password text.

Note Follow these steps when using iTest Selenium session in headless mode, for example, to capture the action of save and file name into the test case:

1. Specify the path to the project to handle exception of conflicts in headless file system.

   For example, store the text response from command as follows:

   ```
   eval file pathToUri [file uriToPath project://iTest_project4/downloads]
   ```

   where `project://iTest_project4/downloads` is the download path

2. Specify a stored variable in the **Download Path** field as it accepts only a file **uri** in headless mode.

   For example, specify a stored variable in the **Download Path** field as **$pathToProject**

After you download, verify the list of files from "**$pathToProject**", for example, use this command: **eval file list $pathToProject**.

addSelection
chooseCancelOnNextConfirmation
chooseOkOnNextConfirmation
click
close
closeWindow
credentials
describe
getInnerText
getSource
go
goBack
goForward
listWindows
mouseOver
open
reload
removeSelection
select
selectWindow
showTable
snapshot

submit

type

waitForPopUp

Captured information about each Selenium page

For each step during test case execution, iTest obtains the following values from the browser's page information. iTest saves the values in the structured data for the response. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze values of interest in the response.

- **Title of the page**
- **URL**
- **Content type** (always set to empty)
- **Encoding**
- **Referring URL**
- **Page load time**
- **Last modification date**
- **Metadata for tags** (built-in browser only)

Selenium Targets and Commands

In Selenium sessions, Actions have ***Targets***. A Target is the control or element that the Action is applied to (the text box to type into, the button or link to click, or the radio button to select, and so on).

Tip You can capture a snapshot of a Selenium page and the XML representation of the HTML code of the page. This feature is useful for documenting the test case and for identifying targets. See the description in "Using the context menu to capture actions on elements" on page 1654.

In the example:

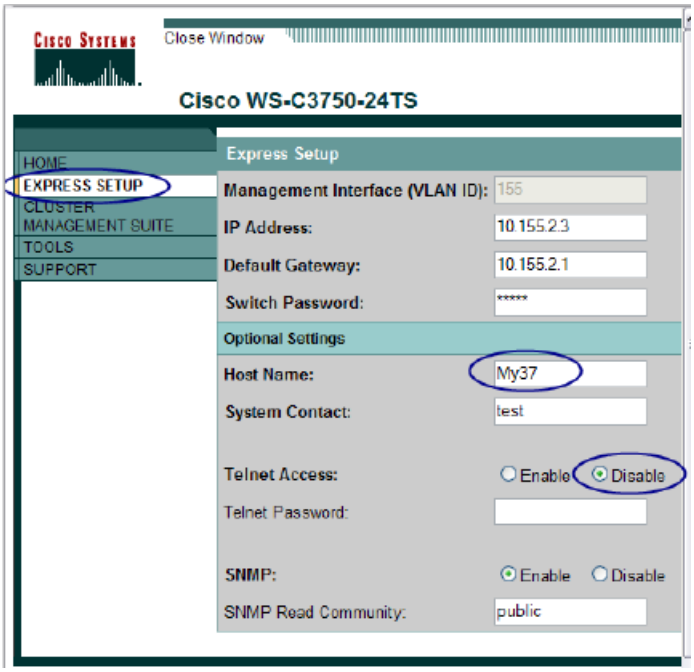
- Clicking a link called EXPRESS SETUP results in a **click** Action with the Target of **link=EXPRESS SETUP**.

  Important To ensure that test cases are as robust as possible, iTest captures the ID for a link (rather than the link itself) whenever a link has an ID. The Target is then the ID of the link.

- Clicking the **Disable** radio button results in a **click** Action with the Target **name=radio_telnetAccess**.

Some Selenium actions include arguments that are captured as iTest ***Commands***. In the example, typing the text **My37** into a text box and then clicking in another part of the page results in a **type** Action with the Target **name=text_sysName** and a command of **My37**.

In the Capture view, the **Description** cell for the captured item displays both the Target and the Command..



CHAPTER 62

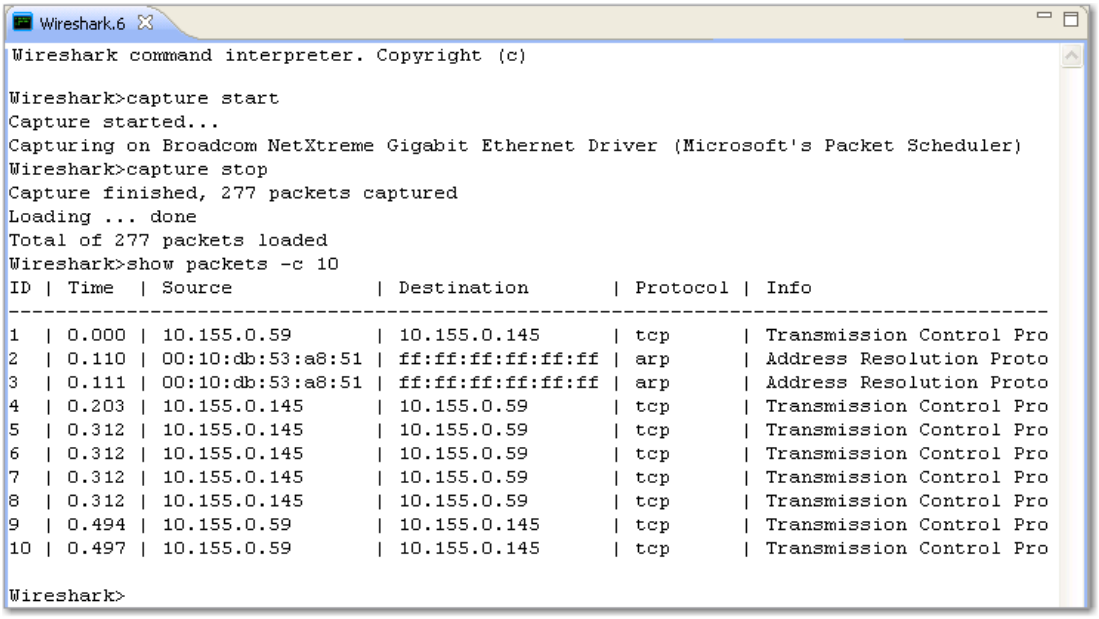
Wireshark sessions

Wireshark session window

Wireshark sessions provide a command line interface for interactively capturing packets from a network interface. For commands that return status and packet data, iTest saves the responses as structured data and generates associated queries to simplify pass/fail analysis.

While the native Wireshark command set is designed to manage the application, iTest uses an abstracted command set that is optimized for automation (for example, commands like waiting for a capture to be completed while filtering the particular data to capture). Session profile properties specify the network interface, a capture filter and display filter, and advanced capture properties for use with **tshark**.

When a Wireshark session is closed, capture stops and all resources used by the session are released.



Analyzing responses

You have the option to store captured packets in memory or write them to a temporary pcap file. You can analyze the content of the PCAP file using iTest or another application (for example Wireshark).

iTest saves all responses to commands as structured data. iTest auto-maps responses (that is, blue boxes appear immediately in the Response view). In addition, iTest auto-generates

appropriate queries, so you can easily work with or analyze the values of interest in the response.

As a result, you do not have to create response maps for the responses and can immediately write analysis rules that use the generated queries.

Microsoft Windows users

Administrative privileges are required to run Wireshark sessions.

About Wireshark commands and responses

- iTest Wireshark sessions support IPv6 packets
- In the Wireshark session window, type **help** to view the list of commands.
- See "Wireshark command set" on page 1222.
- iTest saves responses to some Wireshark commands as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze the values of interest in the response. See "Analysis rules: Validating responses and setting Pass / Fail" on page 641, "Data view" on page 277, and "Structure view" on page 309.

Tips for interactive sessions

- A simple and powerful way to execute a series of commands is to save the command text in a text file (for example, a Notepad text file). Use one command per line. Copy the commands and paste them into an active session at the prompt. The commands execute immediately.
- During an interactive session, click 

in the toolbar and then select the QuickCall from the drop-down list. iTest then executes all of the steps in the QuickCall as if you had typed them yourself. With a single click (typically), you can execute a QuickCall that performs a complex initialization routine or executes a long sequence of related steps.

  QuickCalls will save you a lot of time setting up and tearing down, and, for example, can quickly perform the 20 steps that you usually have to type to bring the device into the proper state for that single crucial test step. See Chapter 9, "QuickCalls: Defining and using a library of custom actions".

Wireshark command set

This topic describes all Wireshark commands that you can submit from iTest.

For the commands noted in the table, iTest saves responses as structured data. In addition, iTest auto-generates appropriate queries, so you can easily work with or analyze the values of interest in the response. See "Analysis rules: Validating responses and setting Pass / Fail" on page 641, "Data view" on page 277, and "Structure view" on page 309.

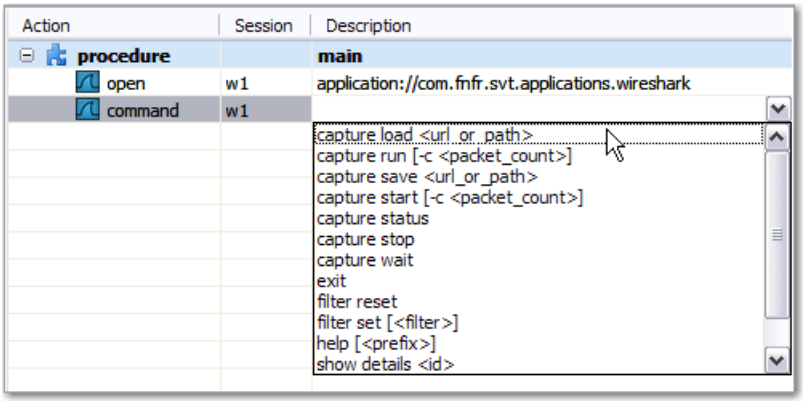
While working in an interactive session

To view the list of commands while working in a session, type **help** at the prompt.

While working on a test case in the Test Case editor

1. For a step in a Wireshark session, in the **Action** cell, select **command**.

2 Select the command from the drop-down list in the **Description** cell. Context help appears while you type.



Command syntax conventions

Convention	Description
boldface	Indicates commands and keywords that are entered literally as shown.
italics	Indicates arguments for which you supply values; in contexts that do not allow italics, arguments are enclosed in angle brackets (< >).
[x]	Keywords or arguments that appear within square brackets are optional.
{x | y | z}	A choice of required keywords (represented by x, y, and z) appears in braces separated by vertical bars. You must select one.
[x {y | z}]	Braces and vertical bars within square brackets indicate a required choice within an optional element. You do not need to select one. If you do, you have some required choices.

Wireshark command set

capture load {*URI*|*path*} -offest <count>	Load captured packets from a file using the **Display** filter. The **Display** filter is specified in the session profile. Use Ctrl-C (or an asynchronous **break** step) to cancel. When you cancel, load stops and some, but not all packets might have been loaded. optional argument **-offest <count>**: Allows you to load packets from a pcap file starting from the specified offset (in packets). For example if **Maximum number of load packets** option in Wireshark preferences is set to 2500 ("Setting preferences for Wireshark sessions" on page 1233) and the **capture load <file> -offset 10000** command is executed, then iTest will load packets with numbers 10000-12500 (a maximum of 2500 packets).
capture run [**-c** *packet_count*]	Start capturing packets using the **Capture** filter and wait for capture to finish. When capture has finished, the packets are loaded using the **Read** filter. Filters are specified in the session profile. Use Ctrl-C (or an asynchronous **break** step) to stop capture. Default: (If **packet_count** is not specified) there is no capture limit
capture save {*URI*|*path*}	Save captured packets to specified location (pcap file). Use Ctrl-C (or an asynchronous **break** step) to cancel.

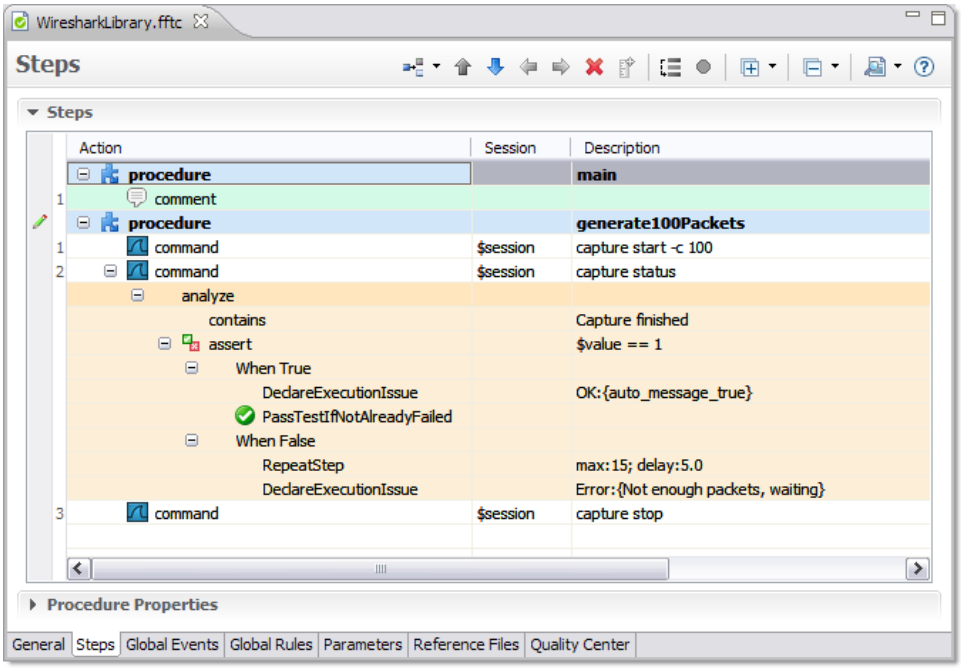
capture start [**-c** *packet_count*]	Start capturing packets in the background. Capture using the **Capture** filter as specified in the session profile. The **capture start** command returns without waiting for capture to complete. If you do not specify the number of packets to capture, capture continues until stopped. Ctrl-C (or an asynchronous **break** step) does not cancel **capture start**. Default: (If **packet_count** is not specified) there is no capture limit **Note** When you execute **capture start**, the packets are not loaded (and therefore cannot be viewed) until after executing a **capture stop** command.
capture-filter set [*filter*]	Set the **Capture** filter controlling which packets will be captured. Use Ctrl-C (or an asynchronous break step) to cancel **capture-filter set**. **Note** The <filter> argument is mandatory in the **capture-filter set** command.
capture-filter reset	Reset the **Capture** filter to make sure that all packets are captured.
capture status	Show the capture status and packet count. After capture has been started, you can detect when capture has finished by using **capture status**. The **capture status** command also shows how many packets have been captured. iTest saves responses as structured data and generates appropriate queries for the status and count.
capture stop	Stop any capture that is in progress and load captured packets into memory using the **Display** filter (the default filter is specified in the session profile).
capture wait	Wait for capture to finish and then load the captured packets using the display filter. Use Ctrl-C (or an asynchronous **break** step) to stop capture. If you want to wait for capture to complete after it has been started, use **capture wait**. Alternatively, use **capture run** to both start capture and to wait for it to complete. The **capture wait** and **run** commands return a response equivalent to **capture end**.
capture statistics	The **capture statistics** command display only PCAP (Packet Capture) loaded through **capture load** command (and not through the **query load** command). Also the **Show decode**, **Show packets, Show details**, and **capture statistics** commands will display packets loaded with the **last capture** command (i.e., capture start-stop/run/load).. **Note** When there are only 1 or 0 packets captured, the **capture statistics** command cannot display the **Avg packets per sec** and **Avg bytes per sec** because the capture duration cannot be measured.
exit	Exit the application
filter reset	Reset the **Display** filter to the value specified in the session profile. Use Ctrl-C (or an asynchronous **break** step) to cancel.
filter set [*filter*]	Update the **Display** filter controlling which packets will be visible. If the optional filter is not specified, then all captured packets are loaded. Use Ctrl-C (or an asynchronous **break** step) to cancel.
help [*prefix*]	Display command help information

query load {*URI*|*path*}	Initializes the specified pcap file to be used by a subsequent **query** command. Uses the **Display** filter. The **Display** filter is specified in the session profile. Use Ctrl-C (or an asynchronous **break** step) to cancel. When you cancel, load stops, but some packets might have been loaded. **Important** **Show decode**, **Show packets, Show details** and **capture statistics** commands will only work after loading a PCAP through **capture load** command (packets loaded with the **query load** command are ignored).
query decode [**-s** *start_ID*] [**-c** *count*] **query details** <*ID*> **query interfaces** **query packets** [**-s** *start_ID*] [**-c** *count*]	Returns information from the pcap file that was initialized by a **query load** command. **Important** You must perform a **query load** command before performing any other **query** command. The **query** command works similarly to the **show** command. Because the **query** action minimizes memory consumption, it is preferred to **show** when the pcap file is too big to fit in memory. **Note** The **query** command must use the **Display** filter (see **filter set** and **filter reset**). See **show decode**, **show details**, **show interfaces**, and **show packets**.
show decode [**-s** *start_ID*] [**-c** *count*]	Returns a list of "decoded" packets received during this session. The decoded packets provide the same values that you see when capturing packets or loading a capture file with tshark.exe or wireshark.exe without iTest. The structure of the response table is similar to the **show packets** command, as illustrated in "Example Wireshark session" on page 1231. In the response table for **show decode**, the Wireshark **Columns** setting determines the response columns. Defaults: start_id=1, count=100 **Note** • When you execute **capture start**, the packets are not loaded (and therefore are not viewable by **show decode**) until after executing a **capture stop** command. • **Show decode** and **capture statistics** commands will display packets loaded with the **last capture** command (i.e., capture start-stop/run/load). **Important** **Show decode** and **capture statistics** commands will only work after loading a PCAP through **capture load** command (packets loaded with the **query load** command are ignored).
show details <*ID*>	Returns the details of selected packet. iTest saves responses as structured data and generates appropriate queries. **Note** • When you execute **capture start**, the packets are not loaded (and therefore are not viewable by **show details**) until after executing a **capture stop** command. • **Show details** and **capture statistics**commands will display packets loaded with the **last capture** command (i.e., capture start-stop/run/load). **Important** **Show details** and **capture statistics** commands will only work after loading a PCAP through **capture load** command (packets loaded with the **query load** command are ignored).
show interfaces	Returns the network interfaces. iTest saves responses as structured data and generates appropriate queries.

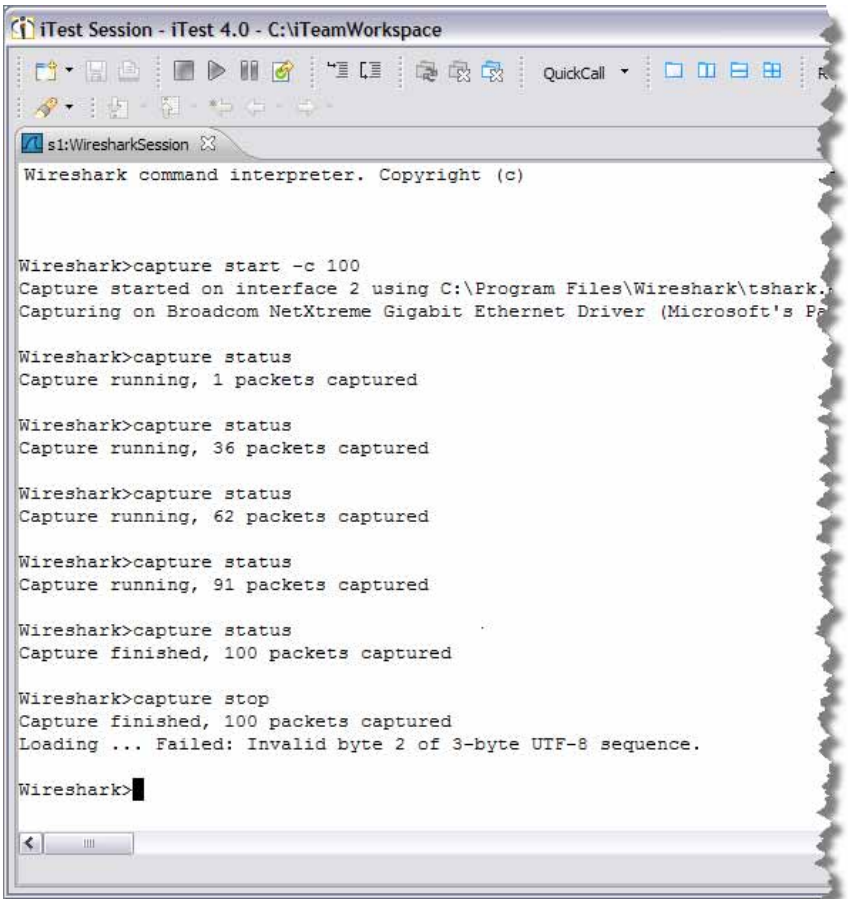
show packets [**-s** *start_ID*] [**-c** *count*]	Lists the packets received during this session. The structure of the response table is illustrated in "Example Wireshark session" on page 1231. iTest saves responses as structured data and generates appropriate queries. Defaults: start_id=1, count=100 **Note** • When you execute **capture start**, the packets are not loaded (and therefore are not viewable by **show packets**) until after executing a **capture stop** command. • **Show packets** and **capture statistics** commands will display packets loaded with the **last capture** command (i.e., capture start-stop/run/load). **Important** **Show packets** and **capture statistics** commands will only work after loading a PCAP through **capture load** command (packets loaded with the **query load** command are ignored).

Example QuickCall that starts and monitors Wireshark capture

In this QuickCall, we perform a **capture start** command and monitor its progress using a **capture status** command.We use a **RepeatStep** action in the analysis rule to execute the **capture status** command every 5 seconds. When the text “capture finished” appears in the response to the **capture status** command, the rule succeeds so that the **capture stop** command can execute, and the QuickCall finishes.



Here is an example execution:



Session profile property settings for Wireshark sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see "Session profiles: Session configuration settings" on page 71. For a detailed description of how iTest uses the settings, see "About property settings" on page 72.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See "About property settings" on page 72.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Wireshark

Capture network interface name	Optional. Specify an interface name to capture packets. Use the **show interfaces** command to see which interfaces can be selected. To select an interface, specify part or all of an interface name or description. Use wildcard "*" characters within the name to find an interface based on a more complex search pattern. Default: <blank>. Capture starts on the first non-loopback interface. Use the **show interfaces** command to see which interfaces can be selected.
Capture filter	Optional. Specify a filter that determines which captured packets are to be loaded into memory. Example: host 10.155.2.3 and tcp See **http://wiki.wireshark.org/CaptureFilters** for a description of the capture filter syntax. Default: <blank> All packets are captured
Display filter	Optional. Specify a filter that determines which captured packets are loaded into memory for display in the responses to **show packets** and **show details**. Example: arp or icmp The Display filter can be more selective than the Capture filter. Refer to http://wiki.wireshark.org/DisplayFilters for a description of Display filter syntax. Default: <blank> All captured or loaded packets are loaded into memory.
Advanced options, capture	Optional. Specify additional tshark options for use during capture. See http://www.wireshark.org/docs/man-pages/tshark.html for a description of the options. Alternatively, enter "tshark help" from a command prompt or shell. **Note** Wireshark already uses the tshark –i, –c, –w and –f options to capture packets and save to a temporary capture file. Arguments that change the output may prevent capture from working. Default: <blank>
Advanced options, display	Optional. Specify additional tshark options for use while loading captured packets into memory. See http://www.wireshark.org/docs/man-pages/tshark.html for a description of the options. Alternatively, enter "tshark help" from a command prompt or shell. Wireshark already uses the tshark –c, –r, –T and –R options to load captured packets. Default: <blank>
Path to tshark	Wireshark sessions use tshark to support network capture and file loading. tshark is installed by the iTest installer. Specify a path here if you did not install tshark in the default location or used a non-standard name for the application. Default: <blank>

Terminal

Local echo	Default: Unchecked Uncheck **Local Echo** to indicate that the device echoes characters typed at the command line. In this case, Spirent iTest ignores echoed characters so that the command text is not added to the echoed response text. For example, if the device does echo and you set Local echo to unchecked, then typing **abc** at the prompt would result in the characters **aabbcc** appearing on the command line. Check **Local Echo** to indicate that the device does not echo typed characters.
Local line editing	Default: Unchecked. Check **Local line editing** to indicate that you may edit line.
Expand all tabs to spaces	Default: Unchecked Check the box to convert each tab character in the response to display 8 space characters in the Response view. This setting can occasionally result in poorly formatted response text in the Response view. Uncheck the box to retain each tab character unchanged.
Scroll to show cursor	Default: Checked While a long command is executing, you might scroll up in the session window to view response data from earlier in the session. When **Scroll to show cursor** is checked, Spirent iTest jumps to the cursor (prompt) when the currently executing command finishes executing.
Terminal string	Default: ANSI Specify the terminal type. Do not change this setting.
Scrollback lines	Default: 10000 Specify the number of command/response lines to display in the session window. These are the lines that you scroll through to view command/response data from earlier in the session.
Encoding	Optional. Specify the encoding type to use to translate bytes into Java characters. You can either type the encoding name into the box or select it from the list. The list includes all encoding types that are supported by the operating system. Default: UTF-8

Terminal > Color

Color scheme	Check **White background** to display black text on a white background. Check **Black background** to display white text on a black background. Default: White background

Terminal > Font

Use standard text font	Check the box to use the fixed width font that appears in terminal windows on the computer that is running iTest. Click **Change** to open the **Fonts** dialog box to specify the font. You can specify the font size, style, type and other effects. Default: Courier New, 10 point, Normal

Example Wireshark session

Start capture, then wait a short while before stopping capture. Capture uses the ‘capture filter’ and other advanced capture properties. When capture is stopped, “Loading...” appears while the temporary capture file is loaded into memory and “done” is appended after.

```
Wireshark>capture start
Capture started...
Capturing on Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet
Scheduler)
Wireshark>capture stop
Capture finished, 277 packets captured
Loading ... done
Total of 277 packets loaded
Wireshark>

// Show the first 10 packets, gives an overview of the main fields (without the
packet details)
// Terminal uses a fixed width of 1000 characters, but this content is wrapped to
fit on your screen. Here’s the actual content:

Wireshark>show packets -c 10
ID | Time  | Source        | Destination     | Protocol | info
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------
-----------------------------------
1  | 0.000 | 10.155.0.59     | 10.155.0.145    | tcp    | Transmission Control
Protocol, Src Port: 1061 (1061), Dst Port: 3389 (3389), Seq: 0, Ack: 0, Len: 0
2  | 0.110 | 00:10:db:53:a8:51 | ff:ff:ff:ff:ff:ff | arp    | Address Resolution
Protocol (request)
3  | 0.111 | 00:10:db:53:a8:51 | ff:ff:ff:ff:ff:ff | arp    | Address Resolution
Protocol (request)
4  | 0.203 | 10.155.0.145    | 10.155.0.59     | tcp    | Transmission Control
Protocol, Src Port: 3389 (3389), Dst Port: 1061 (1061), Seq: 0, Ack: 0, Len: 32
5  | 0.312 | 10.155.0.145    | 10.155.0.59     | tcp    | Transmission Control
Protocol, Src Port: 3389 (3389), Dst Port: 1061 (1061), Seq: 32, Ack: 0, Len: 894
6  | 0.312 | 10.155.0.145    | 10.155.0.59     | tcp    | Transmission Control
Protocol, Src Port: 3389 (3389), Dst Port: 1061 (1061), Seq: 926, Ack: 0, Len: 512
7  | 0.312 | 10.155.0.145    | 10.155.0.59     | tcp    | Transmission Control
Protocol, Src Port: 3389 (3389), Dst Port: 1061 (1061), Seq: 1438, Ack: 0, Len:
247
8  | 0.312 | 10.155.0.145    | 10.155.0.59     | tcp    | Transmission Control
Protocol, Src Port: 3389 (3389), Dst Port: 1061 (1061), Seq: 1685, Ack: 0, Len:
192
9  | 0.494 | 10.155.0.59     | 10.155.0.145    | tcp    | Transmission Control
Protocol, Src Port: 1061 (1061), Dst Port: 3389 (3389), Seq: 0, Ack: 926, Len: 0
10 | 0.497 | 10.155.0.59     | 10.155.0.145    | tcp    | Transmission Control
Protocol, Src Port: 1061 (1061), Dst Port: 3389 (3389), Seq: 0, Ack: 1685, Len: 0

Wireshark>
```

// Now use **capture run** to capture 4 packets and wait for capture to finish (in this case 4 packets are captured but only 2 packets are loaded because of the read filter. Use **filter set** to clear the read filter and show all captured packets)

// The value **4** (in bold) changes from 0 to 4 while the packets are captured.

```
Wireshark>capture run -c 4
Capture started...
Capturing on Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet
Scheduler)
4
Capture finished, 4 packets captured
Loading ... done
Total of 2 packets loaded
Wireshark>filter set
Updating filter ... done
Total of 4 packets reloaded
Wireshark>
```

// Now start capture, optionally execute some other steps (which might generate some packets) and then wait for capture to complete.

```
Wireshark>capture start -c 200
Capture started...
Capturing on Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet
Scheduler)
Wireshark>
Wireshark>
Wireshark>capture wait
200
Capture finished, 200 packets captured
Loading ... done
Total of 200 packets loaded
Wireshark>
```

Setting preferences for Wireshark sessions

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, click **Spirent > Session Types > Wireshark.**

General information on setting and sharing preference settings appears in "Configuring iTest Preferences" on page 861.

Spirent > Session Types > Wireshark

Maximum number of load packets	Default: 2500 **Note** This setting is ignored if you are using tethereal, because tethereal does not support this option. The setting only takes effect if you are using tshark. The load count needs be limited to prevent an Out Of Memory exception when many packets have been captured or loaded. If the number of packets loaded is limited to the maximum packet count then a warning is displayed. You can then change the Display filter setting in the session profile to change the number of packets that will be loaded into memory.

CHAPTER 92

Web Services sessions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Web Services session window

The Web Services session window provides a work surface for composing and submitting Web Service requests.

Any request that you submit in the iTest session is forwarded to the Web Services server. The Web Service performs the action and returns its normal response. You can view the response on the **Response** section of the session window.

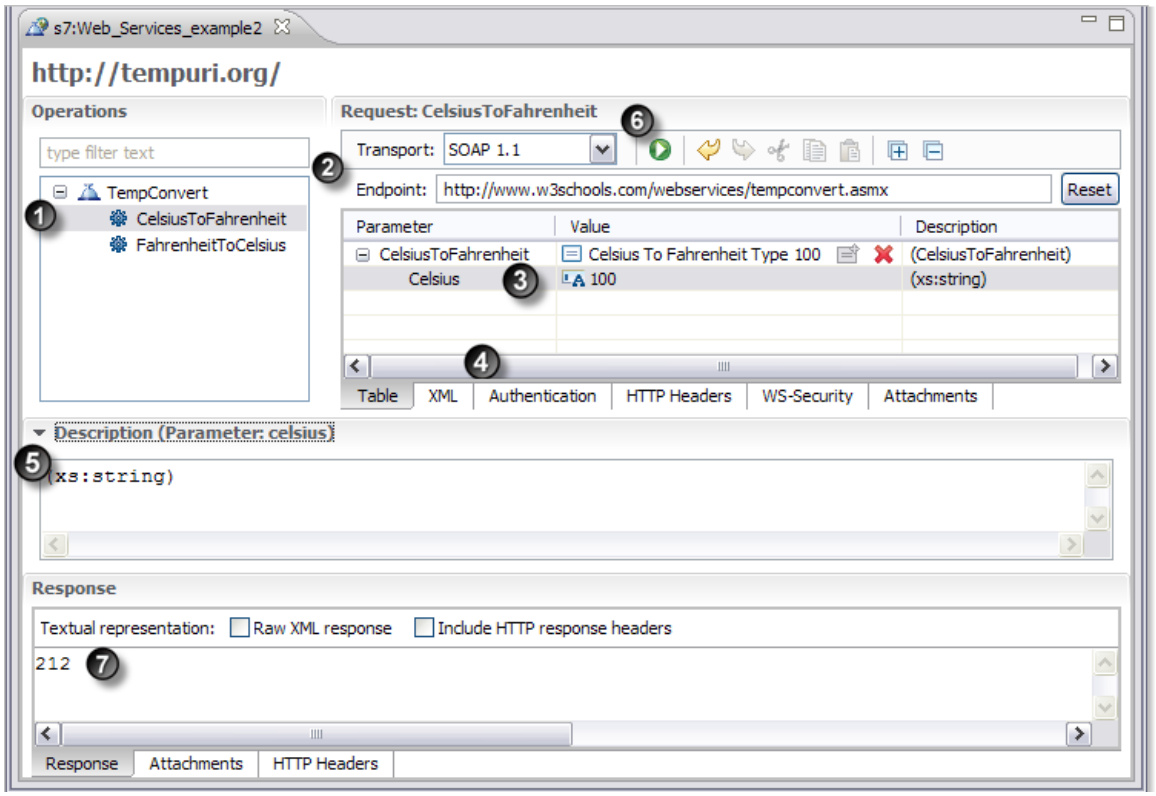
iTest captures all of the actions that you perform in a session and all of the responses. You can use the captured items to create test case steps that interact with the Web Services server.

Example session

In this example Web Services session in iTest, we will prepare and submit a request to convert the temperature of 100 degrees Celsius to the Fahrenheit equivalent.

To prepare to start the session, we first define a session profile that specifies the WSDL file. You have the option to specify an endpoint in the session profile; if you do not specify an endpoint, then iTest uses the endpoint specified in the WSDL file.

When we start the session, iTest opens the Web Services Client session window, obtains the endpoint from the WSDL, and then displays all available operations in the **Operations** list.



1 **Specify the operation** The Operations section displays all services and operations defined in the WSDL file. In our example, we select the **CelsiusToFahrenheit** operation.

Configuring the request

2 **Optional: Specify a different transport protocol or endpoint for the request**

3 **Specify values for parameters** Use the **Table** tab to specify parameter values by typing or pasting. You can add/delete/modify parameters either in the **Table** tab or by typing directly in the **URL** text box. Use the tools to undo/redo changes, cut/copy/paste parameters and parameter groups, and expand/collapse parameter groups.

In our example, we specify the value **100** for the **Celsius** parameter.

Working with elements

- To add a concrete element to an abstract element, click **Create Element** for the abstract element and select the concrete element from the drop-down list.
- To create a new concrete element and insert it into the list of options for an unbounded abstract element, click **Add New Element** and select the concrete element type from the drop-down list.
- Click **Create Element** to create an optional element.
- Click to remove an element.

4 **About the other tabs** •Use the **XML** tab to work with the SOAP envelope for the request — parameters in XML format. The **XML** tab and the **Table** tab are synchronized; changes on one tab are immediately made on the other tab.

- If you specify a new endpoint, you might need to use the **Authentication** page to specify new authentication settings (described in “Authentication” on page 1698).
- Use the **HTTP Headers** page to specify additional headers. If you add a header that is currently in the default header, then the resulting request header will have two values: the value that you specify and the value of the default header. Typically, you specify headers in the session profile to send with all requests.
- Use the **WS-Security** tab to specify the authentication/authorization values that you typically use to access the server. You can configure default settings in the session profile (as described in “WS-Security” on page 1699).
- Use the **Attachments** tab to identify the files to send (as attachments) with the request.

  Click **Add Attachment**  to browse for a file to attach to the request.

  In contrast, you use the session profile to specify the folder for attachments that are returned by the server. See “Returned attachments” on page 1697.

5 **Description** The **Description** section of the window displays the description of the currently selected parameter.

6 **Send the request** Click **Send**  .

When you save an interactive session as a iTest test case, the request is saved as a step.

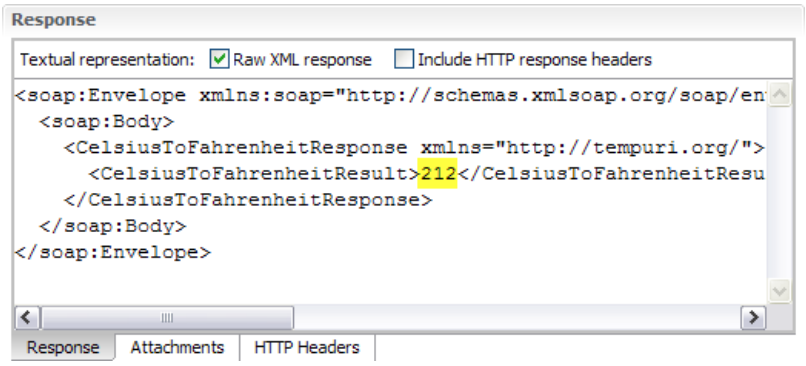
7 **Response** As we expected in our example, the server response is **212** (the equivalent in Fahrenheit of **100** degrees Celsius).

Note iTest auto-maps responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps and can immediately write analysis rules that use the auto-generated queries.

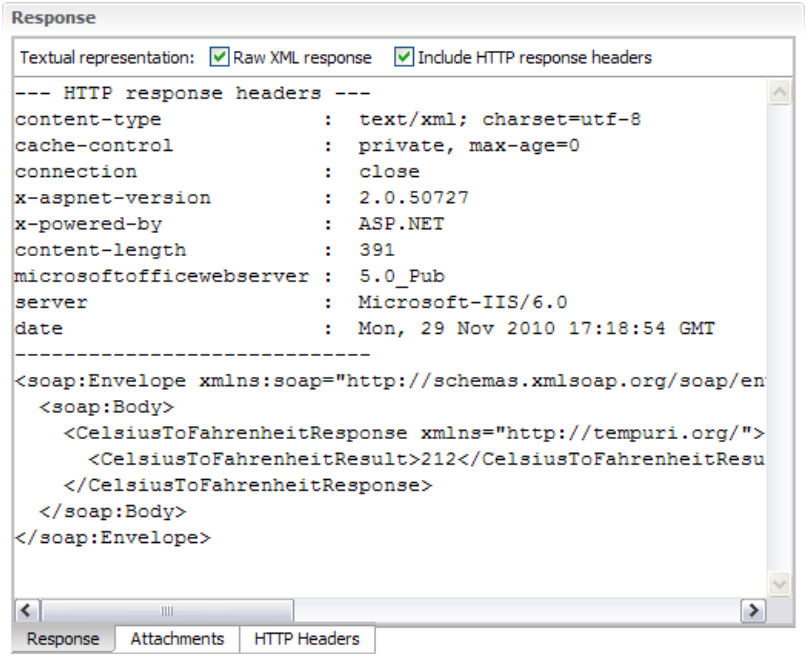
The **Response** section of the session window displays the response from the server in plain text and in several other representations. Regardless of the way you choose to view the response on this page, iTest always captures the full XML response and uses a structured XML version for analysis in iTest.

On the **Response** tab, the default display format is a text representation of the values parsed from the response (a single value, a set of name/value pairs, or a table — the single value **212** in our example). You can view the response in other display formats — specify the format and then resubmit the request:

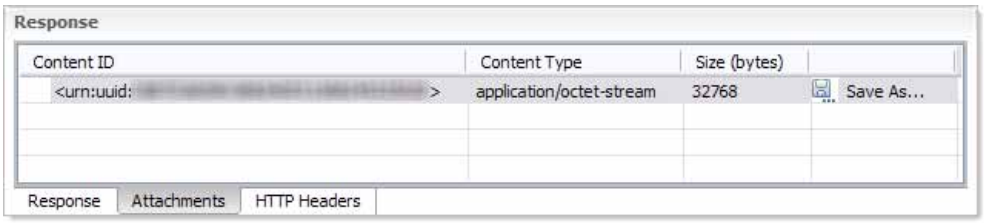
- On the **Response** tab, check **Raw XML response** to view the full XML text of the response — the SOAP envelope. (In the screenshot, we highlight the value that was displayed in the default representation).



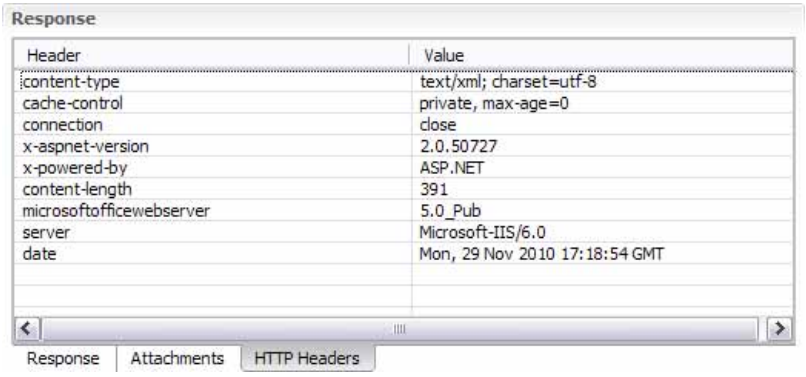
- On the **Response** tab, check **Include HTTP response headers** to display the header text from the response.



- The **Attachments** tab lists attachments returned by the server. Click **Save As** to save the attachment to the file. Graphical file attachments are displayed on the iTest **Images** view during interactive and automated sessions..



- On the **HTTP Headers** tab, the response headers appear in a table.



Note In another example, we might have used a iTest Web Services session to enable a port on a router. After enabling the port, we might open an SSH session with the router to execute a verification step that checks the status of the port that the Web Services server says has been enabled.

Session profile property settings for Web Services sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Web Services properties

Note To use Microsoft Internet Explorer 8, you must turn on Compatibility View. (Click the **Tools** button and then click **Compatibility View**.)

To use a literal IPv6 address in a URL:

- Disable field replacement (substitution) for the property.
- As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html**

WSDL URL	Mandatory. Specify the URL of the WSDL file.
Endpoint URL	Optional. Specify the URL of the endpoint. If you do not specify an endpoint, then iTest uses the endpoint specified in the WSDL file. **Note** You can specify a different URL on the session window while preparing a request.
Accept all cookies	Check the box to accept cookies and save all cookies from the session (as with a typical browser), When you uncheck the box, all attempts by websites to save any data on the client are rejected. This setting is important when you use cookies for storing an authenticated **session ID**. Default: unchecked
Include XML namespace prefixes in response	Check the box to prefix each element in structured responses (including body, siblings, and all parents and grandparents) with a fully qualified namespace. Uncheck the box to specify that responses should *not* include namespace prefixes for body, child, and grandchild elements. Default: checked

Returned attachments

Directory for returned attachments	Specify the file system path for any attachments returned by the server.

Authentication

Authentication	Specify authentication: **None**, **Basic**, or **Secure** Default: None
User	Required if **Authentication** is set to Basic or Secure. Specify the user ID for basic HTTP authentication.
Password	Required if **Authentication** is set to Basic or Secure. Specify the password for basic HTTP authentication.
Key store file	Required if **Authentication** is set to Secure. Specify the path to the client key file (usually has a **.KEYSTORE** file extension). Specify a URI that starts with **file://** or **project://**
Keystore password	Required if **Authentication** is set to Secure. Specify the passphrase to use to access the private key in the key file. The text is masked here and in all locations in which it is used.
Trust any SSL certifiate from the server	Select to ensure that the default certificate from the HTTPS connection is validated. **Important** To use TLS instead of HTML/XML protocol for the Web Services session, you must connect to a secured server (HTTPS), that is select **Secure** authentication and enable **Trust any SSL certificate from the server** **Note** .When you select **Secure** authentication and enable **Trust any SSL certificate from the server**, the **Key store file** and **Keystore password** are not mandatory.

Note When using multiple cipher suites in itest.ini, use comma without any spaces to delimit the entries. If a space is inserted between the cipher and comma (“,”), iTest will fail to interpret the cipher.

Example:

```
Dhttps.cipherSuites=TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
```

The order of the cipher suites in the itest.ini file does not affect the decision of the server.

HTTP Headers

Headers that you specify here are added to each request.

To add a header field, uncheck the **Include inherited values** box and click **Add**  . If you add a header that is currently in the default header, then the resulting request header will have two values: the value that you specify and the value of the default header.

Header	Specify the field name of the header to send with each request
Value	Specify the value for the header.

Proxy

Use proxy	Check the box to use a proxy in sessions. Default: unchecked
Hostname	Specify the hostname or IP address of the proxy
Port	Specify the listening port on the proxy
Username / Password	Specify the user ID and password for authentication.

WS-Security

Specify the values that you typically use to access the server.

Keystore file	
Keystore password	**Add signature**
Signature certificate alias	**Add encryption**
Encryption certificate alias	**Add username token**
Username / Password	**Add timestamp**

CHAPTER 93

XML-RPC sessions

Watch the video

Watch a short video to learn nearly everything you need to know to get started on the Spirent Knowledge Base

Working in the XML-RPC session window

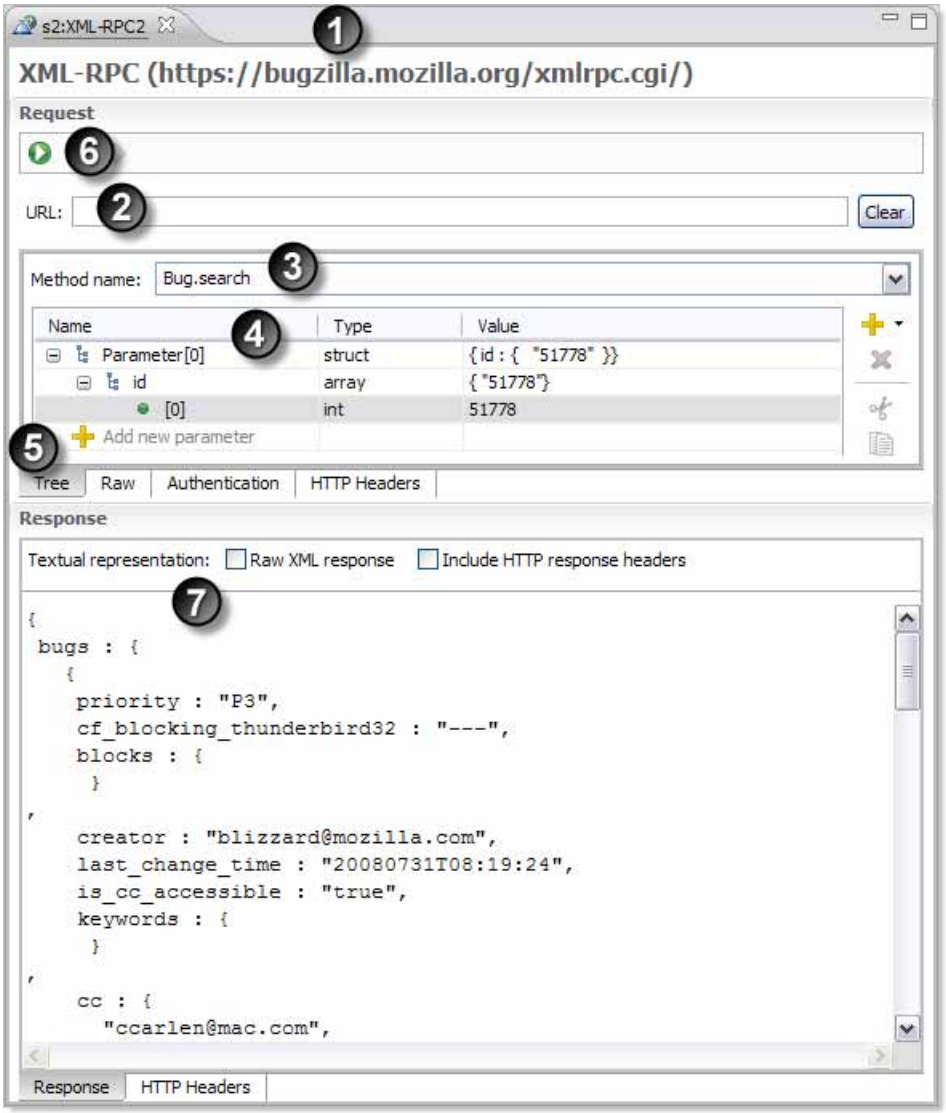
The XML-RPC session window provides a work surface for composing and submitting XML-RPC method calls over HTTP and HTTPS.

Any request that you submit in the iTest session is forwarded to the XML-RPC service. The service performs the action and returns a response. You can view the response in the **Response** section of the XML-RPC session window.

iTest captures all of the actions that you perform in a session and all of the responses. You can use the captured items to create test case steps that interact with the XML-RPC server.

Example XML-RPC session

In this example XML-RPC session in iTest, we prepare and submit an **XML-RPC POST** request to the **Bug.search** method on the **bugzilla.mozilla.org** server. For such a request, the server should respond by returning data for the bug that we specify in the request.



1 **URL of the endpoint** By default, the method call is submitted to the URL that is specified in the session profile that you used to start the session. You can use the URL field 2 to modify the URL as needed. You have the option to specify a URL that is relative to the base URL (like **../path/**) or an absolute URL (like **http://hostname/**).

3 **Specify the HTTP request method** Select the method from the drop-down list of parameters that were specified in the session profile or type a method name.

4 **Specify parameters** Use the **Tree** tab to view the parameters for the XML-RPC service that were defined in the session profile (the default parameters). In addition, you can add, delete, or modify parameters by typing or pasting parameter names and values. You can modify parameters either in the **Tree** tab or by typing directly in the **URL** text box. Use the tools to

add a parameter by selecting the type from the drop-down list, delete , cut/copy/paste , or expand/collapse parameters and undo/redo . In our example, we might verify or modify the default value for integer child of the default array parameter.

To add a child to an **Array** or **Structure** parameter type, select an existing child and click . Alternatively, right-click the parameter name and select **Add Child**.

5 ***About the other tabs***

- The **Raw** tab displays the XML representation of the information on the **Tree** tab (the method and parameter definitions). You can paste, type, or edit the request body. Changes on the **Raw** tab are immediately reflected on the **Tree** tab. Similarly, changes on the **Tree** tab are immediately reflected on the **Raw** tab.
- Use the **Authentication** tab to specify new authentication settings (described in “Authentication” on page 1706).
- The HTTP headers that you specify in the session profile are included with each request URL. Use the **HTTP Headers** tab to add headers to the current request URL.

6 **Send the request** Click **Send** .

Note If we were to save this captured interactive session as a iTest test case, this request would be saved as a step with an **Action** of **XML-RPC POST** and the request (the XML representation of the method and parameter definitions) held in the **Message** property for the step. In test cases, all XML-RPC steps support field replacement (substitution), so you can dynamically modify the request URL at test case runtime. More detail at “XML-RPC test cases” on page 1704.

When you submit the request, you may be prompted for authentication information.

- Provide login/password information if HTTP basic authentication is enabled on the server
- Select a client certificate for HTTPS

7 **Response** As we expected in our example, the server response includes the data for the bug number that we sent in the request.

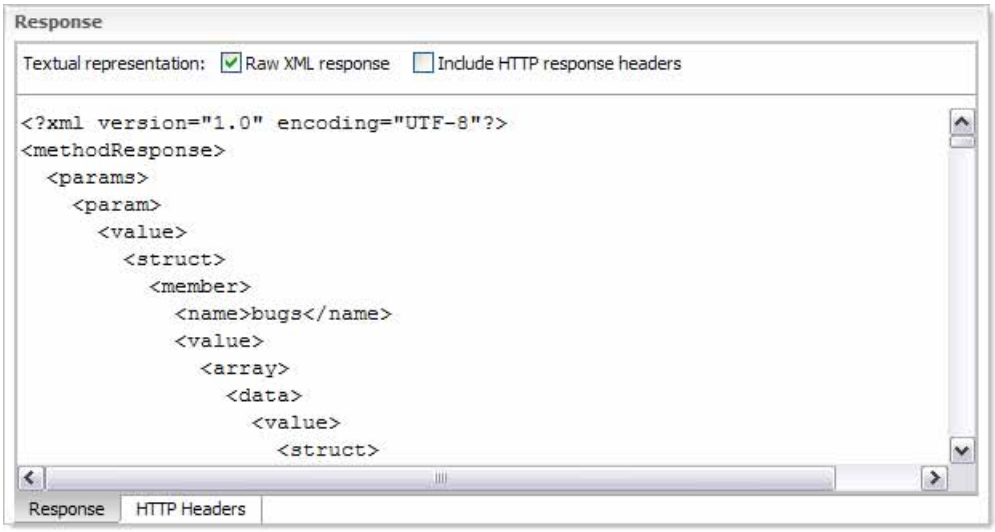
Note iTest auto-maps responses and auto-generates appropriate queries (that is, blue boxes appear immediately in the Response view). As a result, you do not have to create response maps and can immediately write analysis rules that use the auto-generated queries.

The **Response** section displays the response from the server in plain text and in several other representations. Regardless of the way you choose to view the response on this page, iTest always captures the full XML response and uses a structured XML version for analysis in iTest.

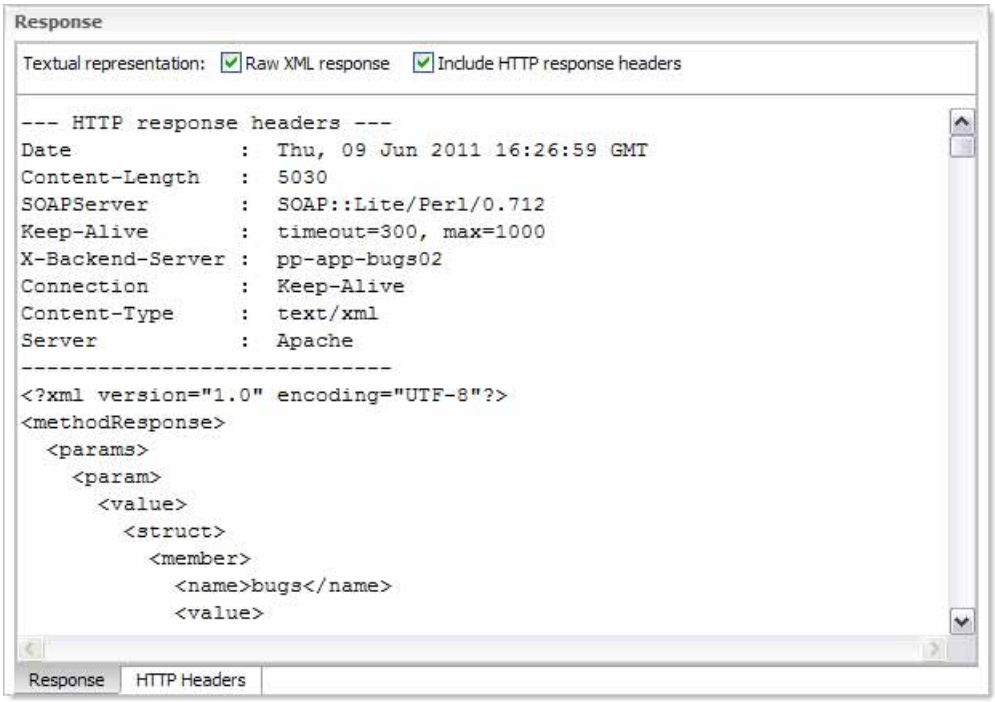
Response formats On the **Response** tab, the default display format (as shown in the example) is a text representation of the values returned in the response (a single value, a set of name/value pairs, or a table). Here are the other formatting options.

Important You must resubmit a request to view the response in a different display format.

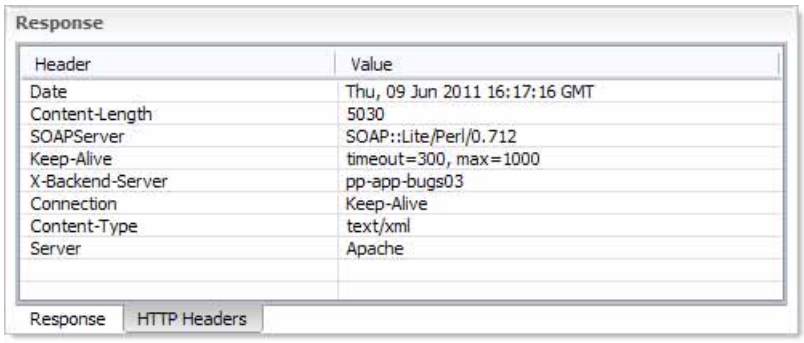
- On the **Response** tab, check **Raw XML response** to view the full XML text of the response



- On the **Response** tab, check **Include HTTP response headers** to display the header text from the response.



- On the **HTTP Headers** tab, the response headers appear in a table

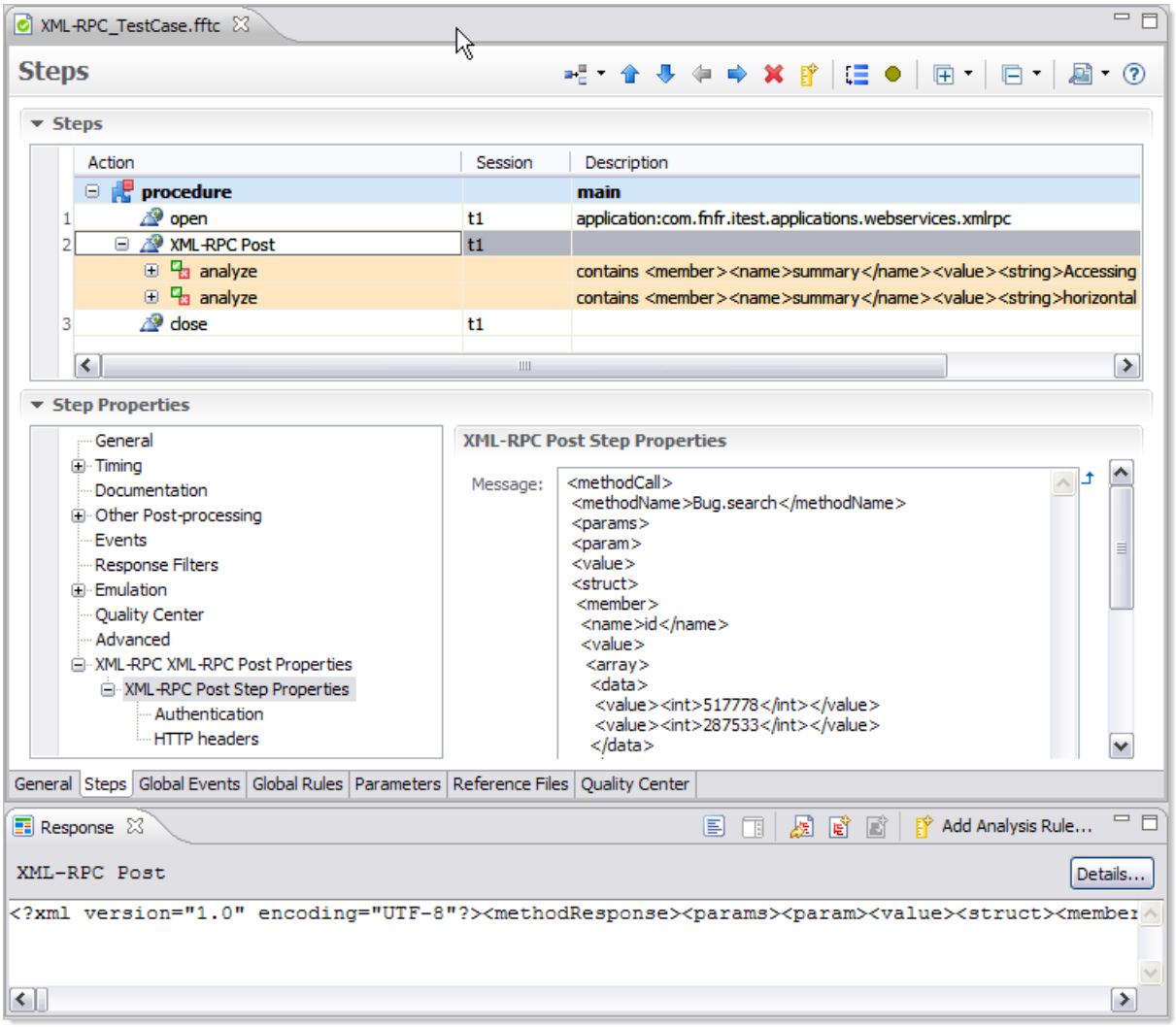


XML-RPC test cases

When you save a captured interactive session as a iTest test case, each request is saved as a step with an **Action** of **XML-RPC POST** and the request (the XML representation of the method and parameter definitions) is held in the **Message** property, as shown here.

In the example, we have saved a session where we submitted the **Bug.search** method one time (and we added two analysis rules).

Tip Remember that the **Raw** tab on the session window also displays the XML representation of the method and parameter definitions. This makes it easy to copy the text from the **Message** property in a test case into the **Raw** tab of a session (or the other way around).



Modifying the request URL or parameters at runtime

In test cases, all XML-RPC steps support field replacement (substitution), so you can dynamically modify the request URL or parameters at test case runtime.

Session profile property settings for XML-RPC sessions

This topic describes the property settings that you can configure for session profiles. For basic information on configuring a session, see “Session profiles: Session configuration settings” on page 120. For a detailed description of how iTest uses the settings, see “About property settings” on page 121.

Note If you have already saved a device or session profile document, then you do not need to configure the settings again. See “About property settings” on page 121.

The first group of properties appears on the **Start a New Session** page (the **Start** tab of the **Session Profile** editor). To access the other settings, click More >>.

Session properties, XML-RPC

URL	URL of the XML-RPC service. To use a literal IPv6 address in a URL: • Disable field replacement (substitution) for the property. • As described in RFC-2732 (http://www.ietf.org/rfc/rfc2732.txt), enclose the literal address in **[ ]** bracket characters. For example, represent **1080:0:0:0:8:800:200C:4171** as **http://[1080:0:0:0:8:800:200C:4171]/index.html** **Note** You can specify a different URL on the session window while preparing a request.

Authentication

Authentication	Specify authentication: **None**, **Basic**, or **Secure** Default: None
User	Required if **Authentication** is set to **Basic** or **Secure**. Specify the user ID for basic HTTP authentication.
Password	Required if **Authentication** is set to **Basic** or **Secure**. Specify the password for basic HTTP authentication.
Keystore file	Required if **Authentication** is set to **Secure**. Specify the path to the client key file. Specify a URI that starts with **file://** or **project://**
Keystore password	Optional if **Authentication** is set to **Secure**. Specify the password to use to access the private key in the key file. The text is masked here and in all locations in which it is used.

HTTP Headers

Headers that you specify here are added to each request URL.

To add a header field (an **HTTPHeaderEntry**), uncheck the **Include inherited values** box and click **Add** .

Header	Specify the field name of the header to send with each request
Value	Specify the value for the header.

Method Names

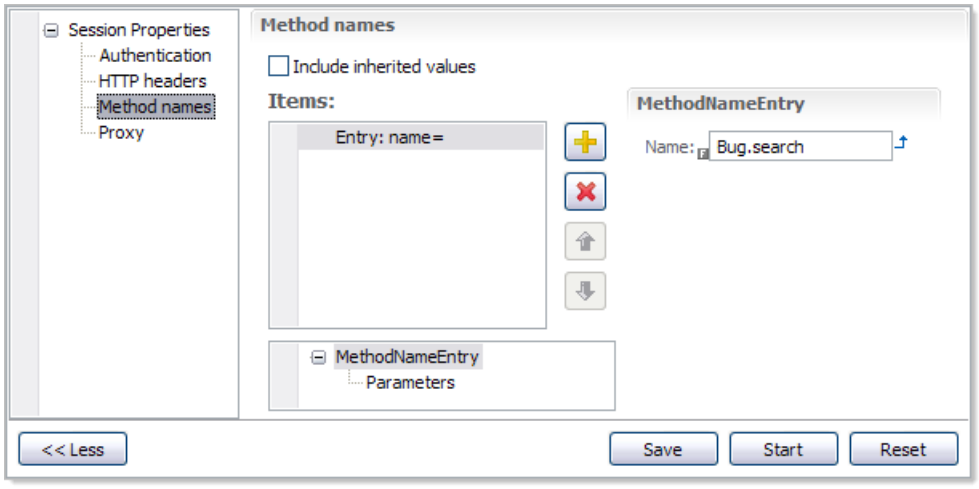
Methods that you specify here are added to each request URL.

During an interactive session, you can edit methods that you specified here or specify additional methods before submitting the request.

To specify a method (a MethodNameEntry) and associated parameters

1 Uncheck the **Include inherited values** box and click **Add** .

2 Type the method name into the **Name** text box.



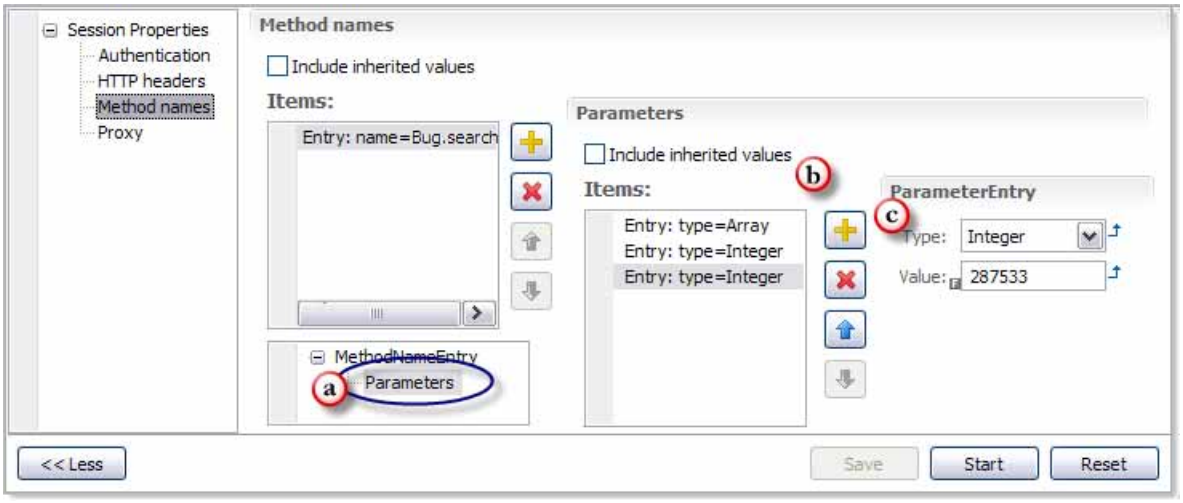
3 To add a parameter value:

a Click **Parameters** in the tree.

b Uncheck the **Include inherited values** box and click **Add** .

c Select a **Type** and specify a **Value**. In the example, we specified an Array parameter and then added two Integer values to the array. The **Base64** type enables you to browse for a file.

Note On this page, you cannot add children to **Array** or **Structure** parameter types. You add children using the session page.



4 To specify another method, click the text **MethodNameEntry** in the tree and repeat the process.

Tip Remember that during an interactive session, you can add, delete, or modify these parameter settings, as described in “Specify parameters” on page 1701.

Proxy

Use proxy	Check the box to use a proxy in sessions. Default: unchecked
Hostname	Specify the hostname or IP address of the proxy
Port	Specify the listening port on the proxy
Username / Password	Specify the user ID and password for authentication.

CHAPTER 64

Debug Velocity Drivers and Executions

Overview

iTest provides a way to enable debugging of automation assets being run from Velocity. This allows you to fix any issues with drivers, automation tasks, and test cases being developed. iTest opens the test cases being debugged in the i**Test Debugging** perspective. You may view multiple windows—Velocity Explorer with test artefacts, responses, structures, and breakpoints of Velocity test executions (in the i**Test Debugging** perspective). The following topics are included:

Configuring iTest Gui as an Agent

To view or edit preferences, click **Window > Preferences.** On the **Preferences** page, do the following:

Configure Velocity preferences

Click **Spirent > Velocity** and configure the Velocity server FQDN and access details.

Server URL	Specify the FDQN of the host where Velocity is running.
User name and Password	Optional: Specify the default username and password used to log in when iTest starts.
Sync interval (sec)	Velocity periodically checks for changes on the Velocity server (topologies, resources, and reservations) to ensure that the data is always "in sync" with iTest. Specify the time interval between data refreshes. Default: 30 seconds
Login mode	Select to indicate the login mode to Velocity when iTest starts: • **Automatically log me in:** Select to ensure that iTest automatically logs you in when the correct server URL and login credentials are provided. • **Prompt me to login every time:** Select to ensure that iTest prompts you to login to Velocity, at startup. • **Do not auto-login at startup:** Select to ensure that iTest does not automatically log into Velocity at startup, even when the correct Velocity URL and login credentials are entered.

Configure Connection

Click **Spirent>Velocity > Connection** to specify the following settings when connecting to velocity. For example, when connecting to a Velocity instance in a remote lab, to taking into account any latency, specify the time to wait before iTest displays an error.

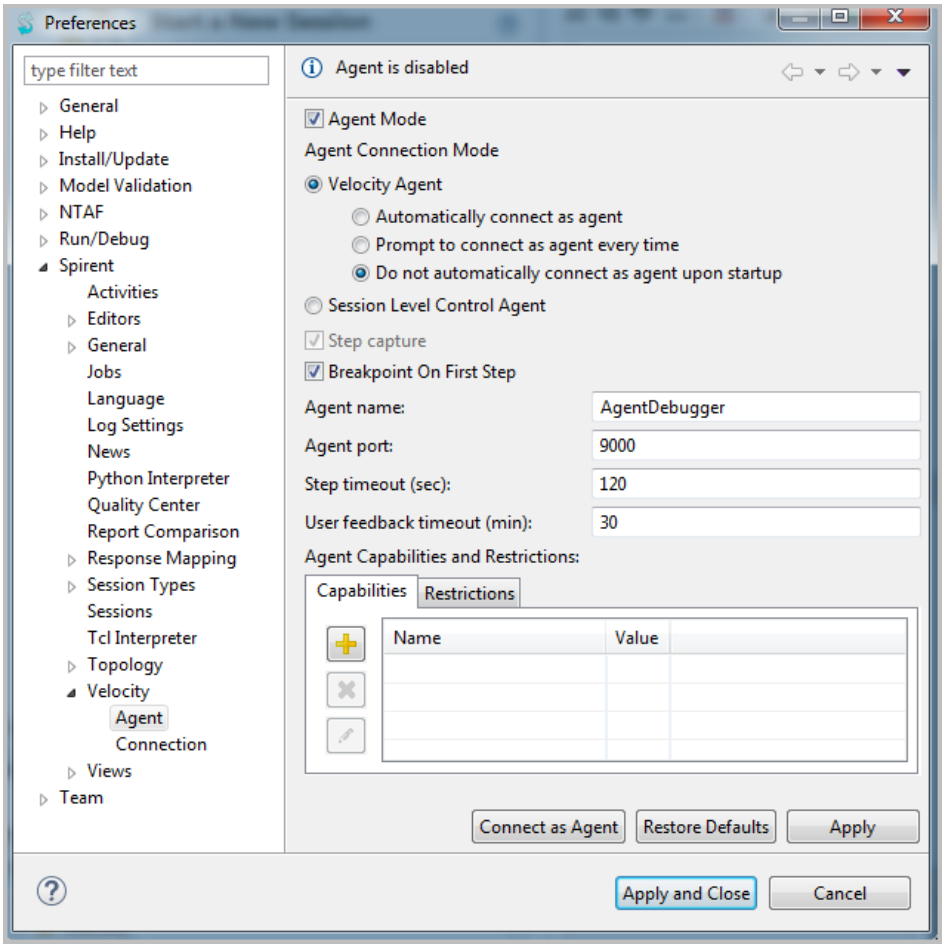
- Socket time out (sec): Defaut: 4 s. Indicates the time to wait until network establishes connection.
- Connection timeout (sec): Default: 10. Indicates the time to wait before Velocity connection fails.
- Connection request timeout (sec): Default 20. Indicates the time to wait before request for connection fails.

Configure Agent Mode

Click **Spirent > Velocity > Agent:** enable **Agent mode** and complete as described below.

Agent Mode	Select to enable Agent Mode
Agent Connection Mode	Select an option below to indicate how you wish to be connected to the Agent. • **Automatically connect as Agent:** Select to ensure that iTest automatically connects as an Agent at startup. • **Prompt to connect as Agent every time:** Select to ensure that iTest prompts you at startup to confirm whether iTest should connect to Velocity Server as Agent. (Default) • **Do not automatically connect as Agent at startup:** Select to ensure that iTest does not automatically connect to Velocity Server as an Agent at startup. **Note** The **Session Level Control Agent** is disabled when **Agent Connection Mode** is selected.
Session Level Control (SLC) Agent	N/A in Agent Connection Mode. See "Configure Listening Mode (Session Level Control Agent)" on page 1169 (Chapter 59, "Python Automation Library"). This option enables the listening mode. iTest GUI does not connect to Velocity as an Agent, but waits for the incoming connections instead. **Note** The **Agent Connection Mode** is disabled when **Session Level Control Agent** is selected. iTest either connects as an Agent or acts as a **Session Level Agent** server. In **Session Level Agent** mode, the Agent listens for SLC connection (iTest GUI waits for connections) and the Python Automation Library connects to iTest GUI when available.
Step capture	The Step capture option becomes available only when the **Session Level Control Agent** is selected. • Selected: (Default), the session actions performed by the **Python Automation Library** on this iTest GUI instance are captured. • Not selected: the session actions performed by the **Python Automation Library**library on this iTest GUI instance are *not* captured.
Breakpoint at first step	Default: Selected Select to insert breakpoint at first step, if desired. **Note** Breakpoints tell iTest to pause execution. See Chapter 20, "Debugging Test Cases"
Agent Name:	Enter a name for the Agent.
Port	Indicates the port used by the Agent during execution. • **Port for 6.1 (and earlier) Driver Agent:** Default: 9001 • **Port for Test Agent:** Default: 9000 • **Port for Session Level Control Agent**: Default 9005

Step timeout (sec)	Specify a time limit in seconds to apply to all steps in the test cases using this Agent. if a step is not completed within the specified time limit, iTest stops the test execution. **Note** This setting overrides the Default step timeout property specified on the General page of the Test Case editor.
User feedback timeout (min)	Default: 30 iTest pauses execution for a period specified in **User feedback timeout (min)** and prompts you to confirm where you wish to continue execution. This is to prevent deadlocks and endless executions. Velocity displays a message saying that the Agent Execution will automatically be aborted in the species number of minutes, displays a count-down time and asks you whether to **Continue Execution** or **Abort**.
Agent Capabilities and Restrictions	Specify a unique capability/restriction to limit the volume of test executions. **Note** The name/value pair you enter is not case sensitive. iTest converts the name and value pair to lower case. **Capabilities:** Enter a name value pair for Agent capabilities. Example: os.type, win 32 os.type, linux **Restrictions:** Enter a name value pair of Agent capabilities exclusive for your use. Indicates, that the agent (name, value pair) are used to limit the volume of test executions. For example, when debugging a test case or a driver, you would ensure that the agent capabilities are restricted for your executions only.
Apply and close	Click to apply settings and **Connect as Agent** and close the window.
Restore Defaults **Apply**	**Restore default:** Click to discard all the changes made and reset to the default values. **Apply:** Click to apply the changes made.



Note General information on setting and sharing preference settings appears in Chapter 41, "Configuring iTest Preferences".

Debugging Driver Agent

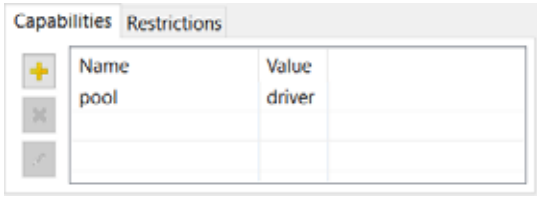
Important The steps in this section are applicable for debugging driver agents with Velocity 7.x and later. For debugging Driver Agent for Velocity 6.1 see *iTest 6.1 UserGuide*.

To ensure that the iTest UI connects to Velocity as a driver Agent, make sure you have set up the **Window > Preferences** correctly as in "Configuring iTest Gui as an Agent" on page 1263.

1 In the **iTest > Windows > Preferences** dialog:.

Enter a name for the Agent in **Agent Name**, example: **AgentDegugger**

Enter **pool/driver** as a Name/Value pair in the **Capabilities** tab.

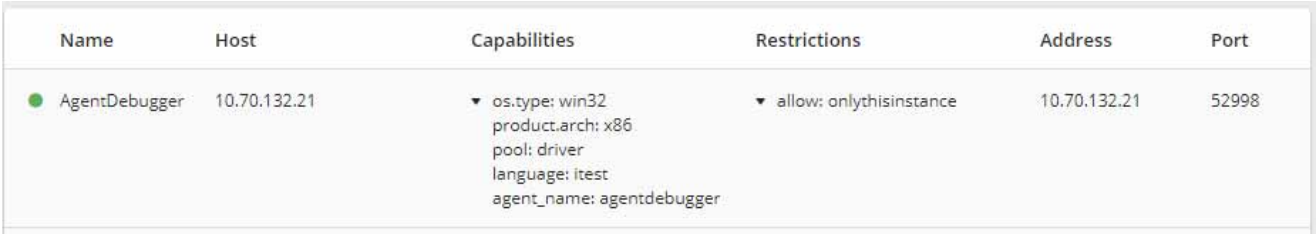


In the **Requirements** tab, enter a Name/Value pair with unique values. This is so that your driver will only execute on the agent you are debugging with.

Example:

Click **Apply** to see for the changes to be affected).

Tip In Velocity UI (**REPORTS>Velocity Agents**), see that the Agent is listed under **Name** section. Also that the **Capabilities** and **Restrictions** match your environment.



2 Find the driver to debug:

a In Velocity, go to Library > Drivers and download the driver you wish to debug (e.g., Ping Driver 1.1.0).

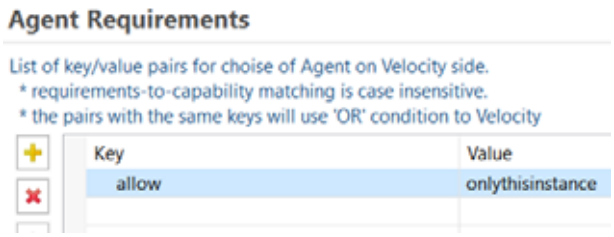
b In iTest, import the driver files from the downloaded archive file. (see “iTest Import wizard” onpage 1727, in Chapter 97, “Wizards and Dialog boxes”).

c You may want to rename the project (so that you don’t break or interfere with existing drivers).

d In iTest, open the driver in Test Case editor and add the following on the **Requirements** tab as capabilities name/value pair with unique values.

Important The **Agent Requirements** must match the **Restrictions** of the driver in Velocity. This is so that your agent will only execute the driver you are debugging and not execute other drivers.

Example:



Save the test case.

e In iTest, export the driver test case and the manifest project as an iTar file. (see "Exporting iTest projects as itar files" on page 815, in Chapter 37, "Sharing iTest Resources").

3 In Velocity UI (**LIBRARY>Drivers**):

a Upload the new driver to Velocity. For a new driver, click the **Add** button. For and existing driver, select the driver and click **Edit** button.

b Create a resource that uses the new driver and then discover it (click **Discover** on the **INVENTORY>Resource** page).

4 For example, if you choose to discover some basic resource from Velocity, the corresponding driver from Velocity will open, in iTest, in the **iTest Debugging** perspective for investigation.

Click the items in the Velocity Explorer to view. Edit as needed. Save files and export to back to Velocity

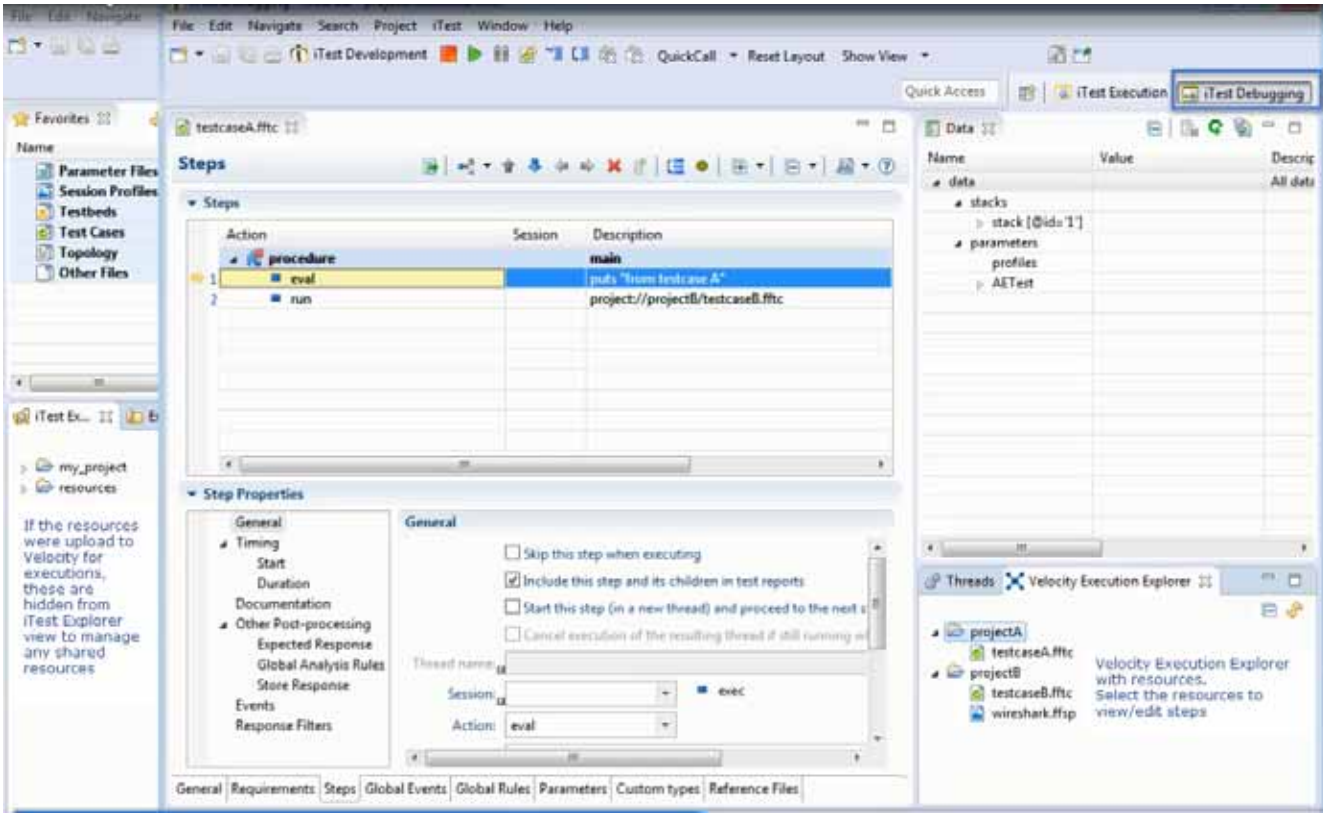
Note For more on debugging any iTest Test case. See Chapter 20, "Debugging Test Cases".

Debugging Velocity Test Executions using iTest GUI

To ensure that the iTest UI connects to Velocity as a Test Agent, make sure you have set up the **Window > Preferences** correctly as in "Configure Agent Mode" on page 1264.

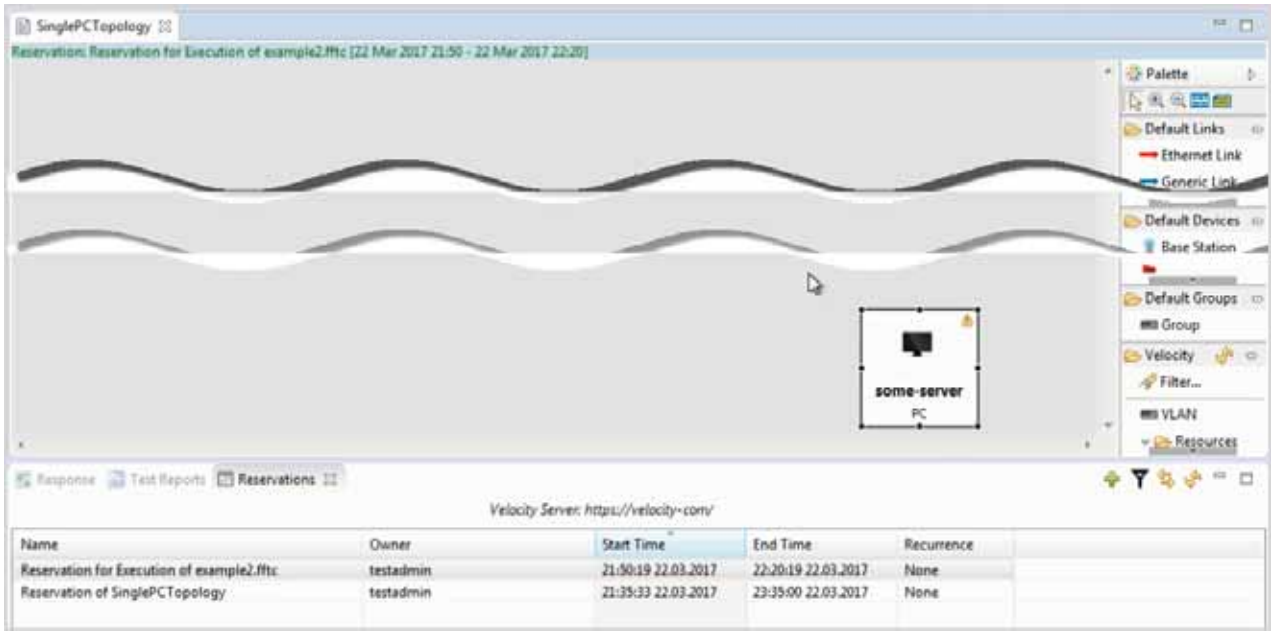
Note Ensure that the **Driver Executions only** option is not selected and, agent capabilities and restrictions are specified. The test agent and capabilities will be listed in **Velocity > Reports> Velocity Agents**. The agent will not be listed if you specify restrictions.

In Velocity, select the Automation Asset to execute, when the execution starts, go to the iTest GUI and notice the following.



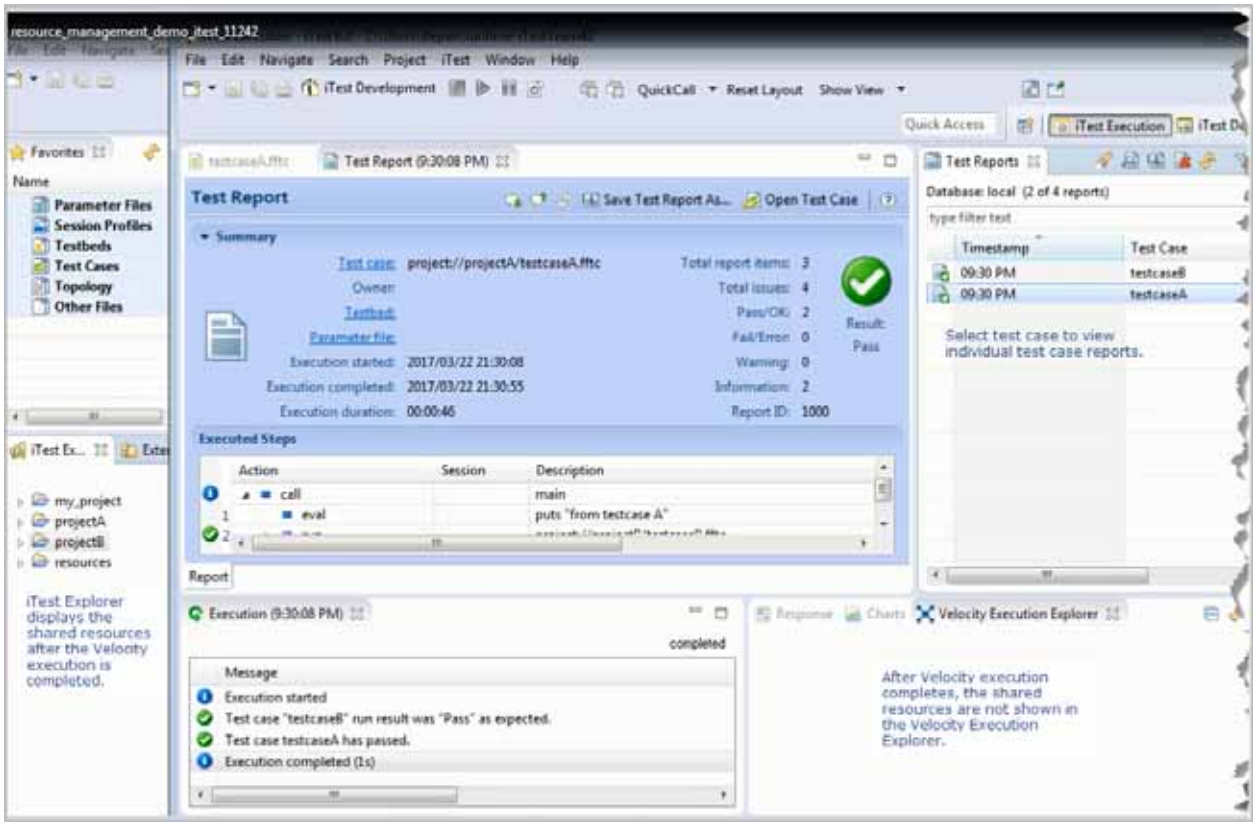
- Velocity Execution Explorer, displays the files included in the iTar. If you share resource between iTest and Velocity, these resources are hidden in the iTest Explorer until the Velocity execution completes.

If the Velocity execution included a Topology, you may view it in iTest GUI from the Reservation tab.



- If you have configured **Breakpoint at first step** option, iTest pauses execution until you click to continue execution.
- If a test case step execution is not completed within the time limit specified in **A Step timeout (sec)**, iTest stops the test case.
- iTest pauses execution for a period specified in **User feedback timeout (min)**. iTest displays a message saying that the Agent Execution will automatically be aborted in the specified number of minutes, displays a count-down time and asks you whether to Continue Execution or Abort. This prevents deadlocks and endless executions.

- After execution completes, the full test reports are stored in derby or external database.



Once the execution completes, the shared resources are displayed in iTest Explorer and the Velocity Execution Explorer displays blank. Select the test case from the Test Reports tab to view details.

Debugging Driver Agents in the iTest 7.0 and Velocity 7.0

CHAPTER 67

Using Git in iTest

Overview

iTest integrates Eclipse EGit plugin, which allows you to use Git source control from iTest. This chapter provides instructions on using EGit from within iTest.

The instructions assumes the following:

- Groups within your organization want to work on a project using EGit to develop and maintain test cases in a single location (the **master branch**).
- Users will have their own local repository (a copy of the test cases code including all the source control relevant information).
- Each user will receive changes and send changes using a remote repository at GitHub.

For details about EGit see http://wiki.eclipse.org/EGit/User_Guide.

This chapter includes the following section.

Note Set Git preferences in iTest as described in "Setting preferences for Git" on page 1318 first and then perform the rest of the tasks described in this chapter.

Setting up Git repository in iTest

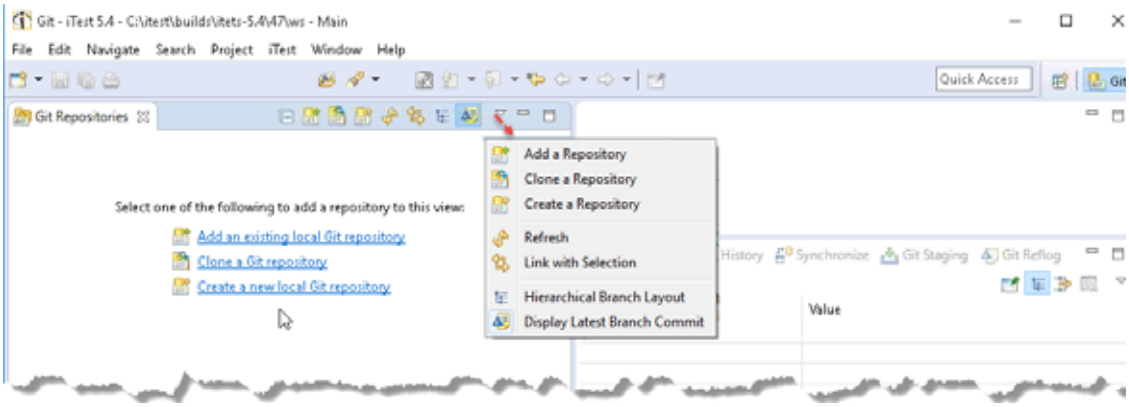
Follow these instructions in iTest.

Step 1 Setup Local Repository

1 Click Open Perspective ( ) at top right-hand side. Then Click **Git** in the **Open Perspective** dialog.

The **Git Repositories** page opens.

Note You may use the menu options on the top of the **Git Repositories** page as required.

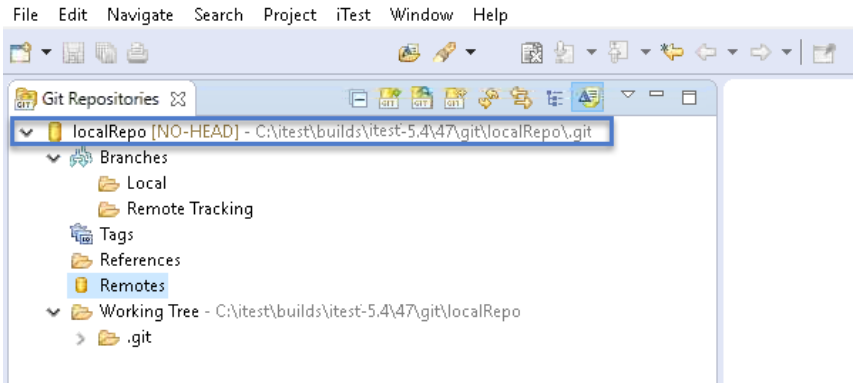


2 Select **Create a new local Git repository** the repository to add to your current view.

Create a New Git Repository dialog opens.

3 Click **Browse**, navigate and choose the directory for your new local repository. For example: (example: **c:\itest\builds\itest-5.4\47\git\localRepo**) Click **Finish.**

A Local Git Repository is created at the specified location.

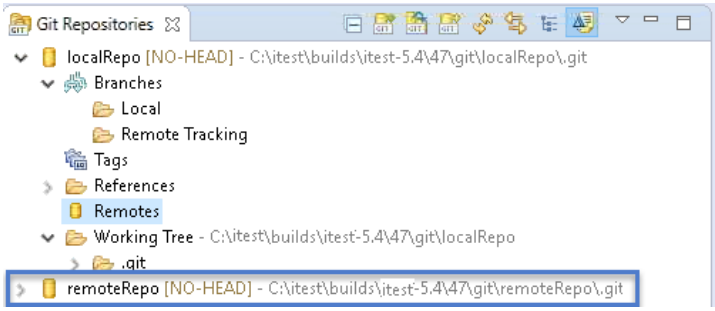


Step 2 Setup Remote Repository (Locally)

1 Select **Create a new local Git repository** from the menu (at the top of the **Git Repository** page) to be added to your current view.

Create a New Git Repository dialog opens.

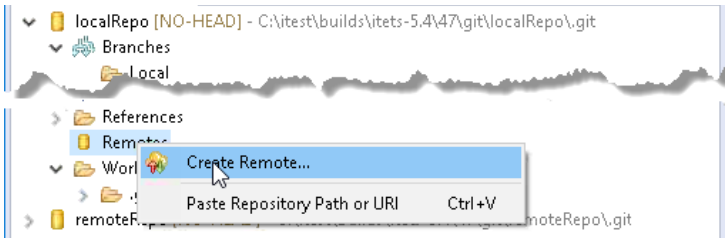
2 Enter the remote directory path (example: **c:\itest\builds\itest-5.4\47\git\remoteRepo**) or navigate to the location. Click **Finish**.



A Remote Git Repository is created at the specified location.

Step 3 Create Remote repository to push or fetch files

1 Right-Click on **Remote** under the **localRepo** and the **New Remote** dialog opens.

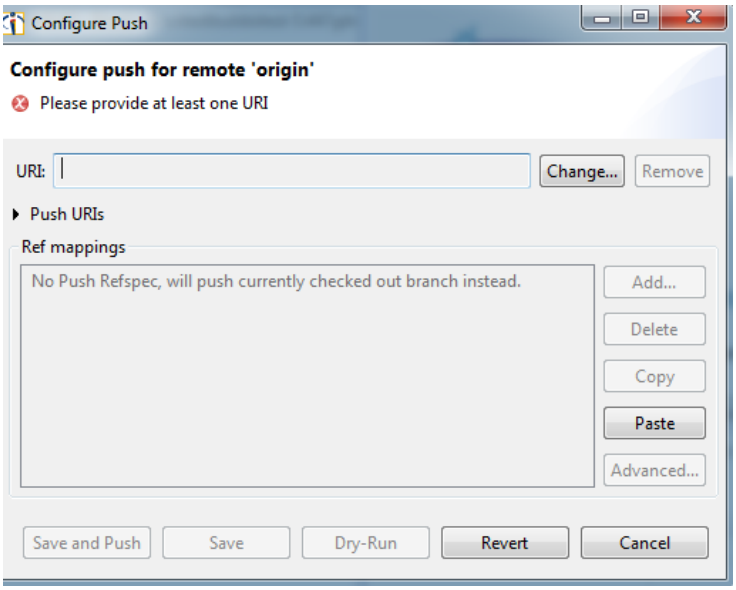


2 Select the type of remote location you wish to create on the New Remote dialog. Enter/select the following:

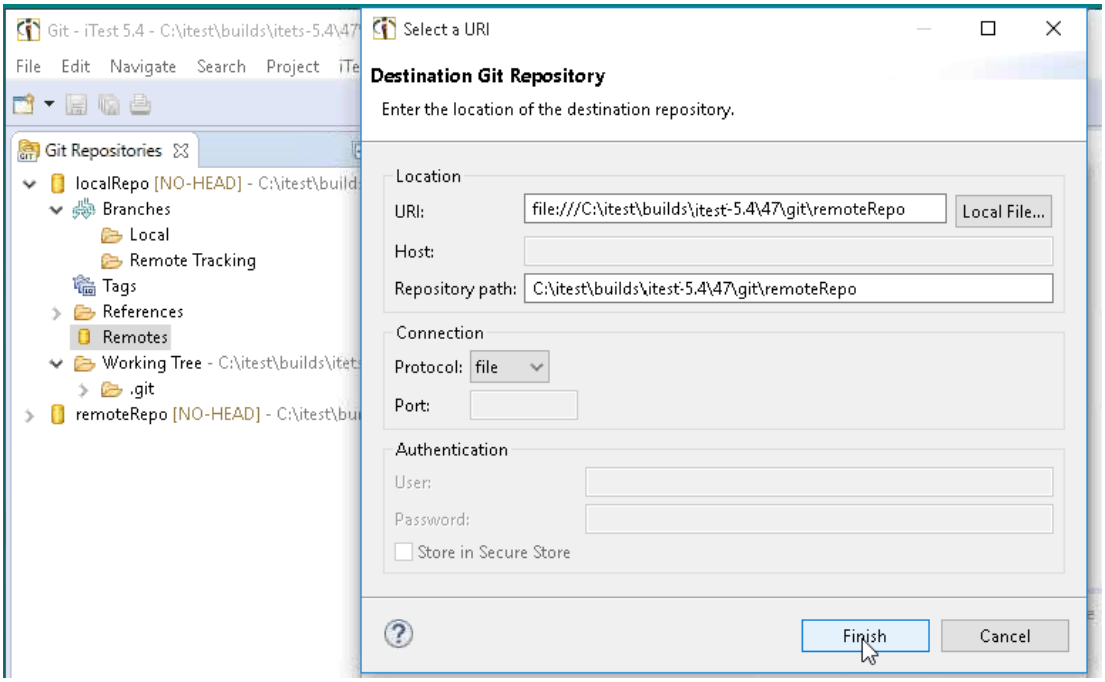
Remote name:	Default: **Origin** Enter the remote repository name (this name appears on the remote configuration dialogs).
Configure push	Select to configure the new remote location for **push** operation (selected by default).
Configure fetch	Select to configure the new remote location for **fetch** operation.

It is mandatory that you configure the new remote location for either fetch or push operations. You may add configuration for the other directory at a later time.

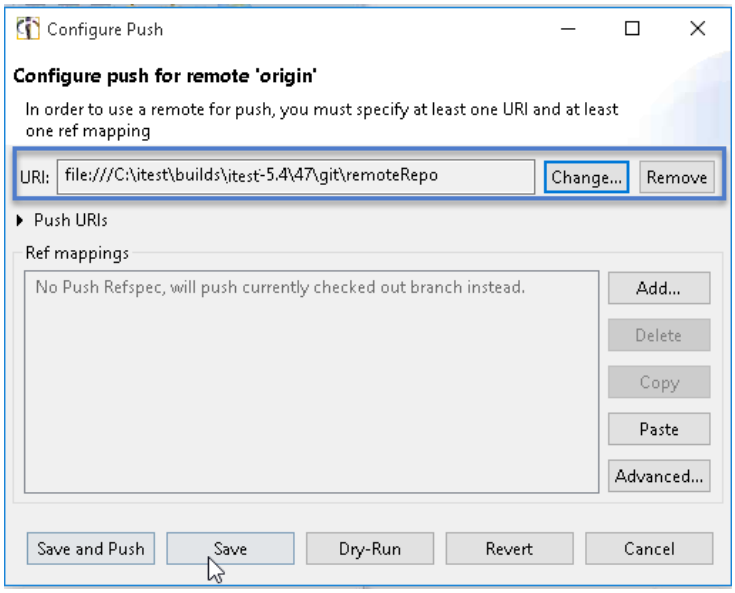
3 Select **push** and click **OK** and **Configure push for remote 'Origin'** dialog displays.



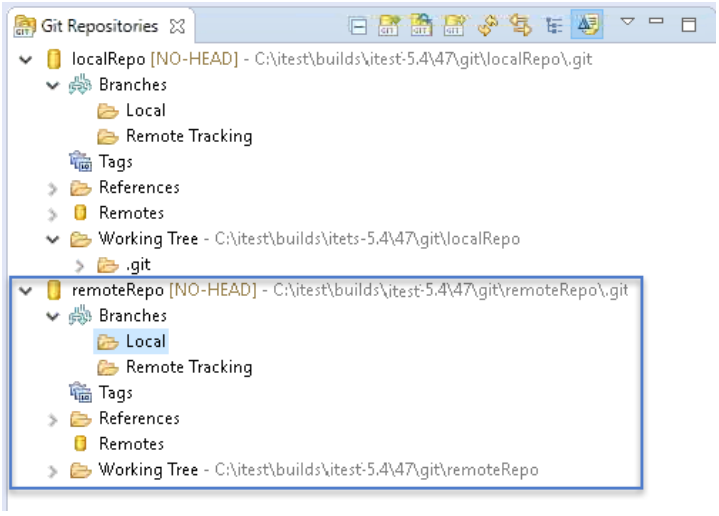
4 It is mandatory to provide a valid URI. Click change and when the **Destination Git Repository** dialog displays, browse to the location and provide the correct destination **URI** and **Repository path** as illustrated.



5 Click Finish and the **Configure push for remote 'Origin'** displays populated with the destination URI you specified on the **Destination Git Repository** dialog.



6 Click **Save** and verify that the remote repository is created as required.



After setting up the Git Repository in iTest, the next step is to add your project folder to Git repository. See section "Add your iTest Project Folders to Git" on page 1308 below.

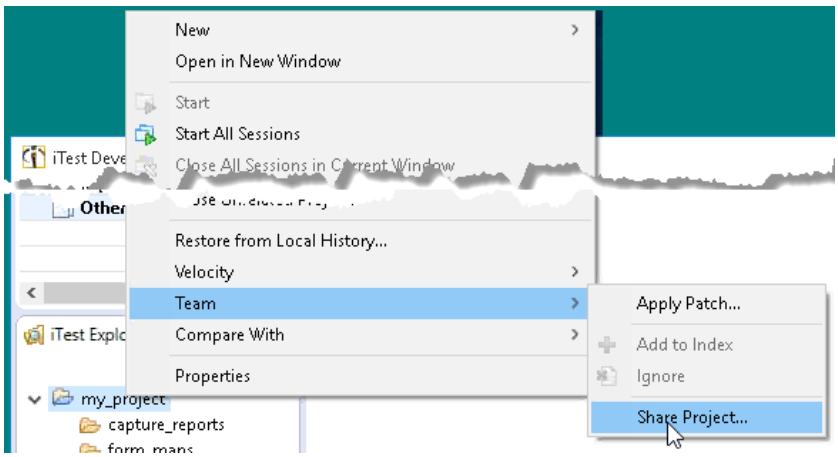
Add your iTest Project Folders to Git

This section provides instruction to add your iTest project folders to Git. This involves adding the project to the local repository and then uploading (push) to the remote repository.

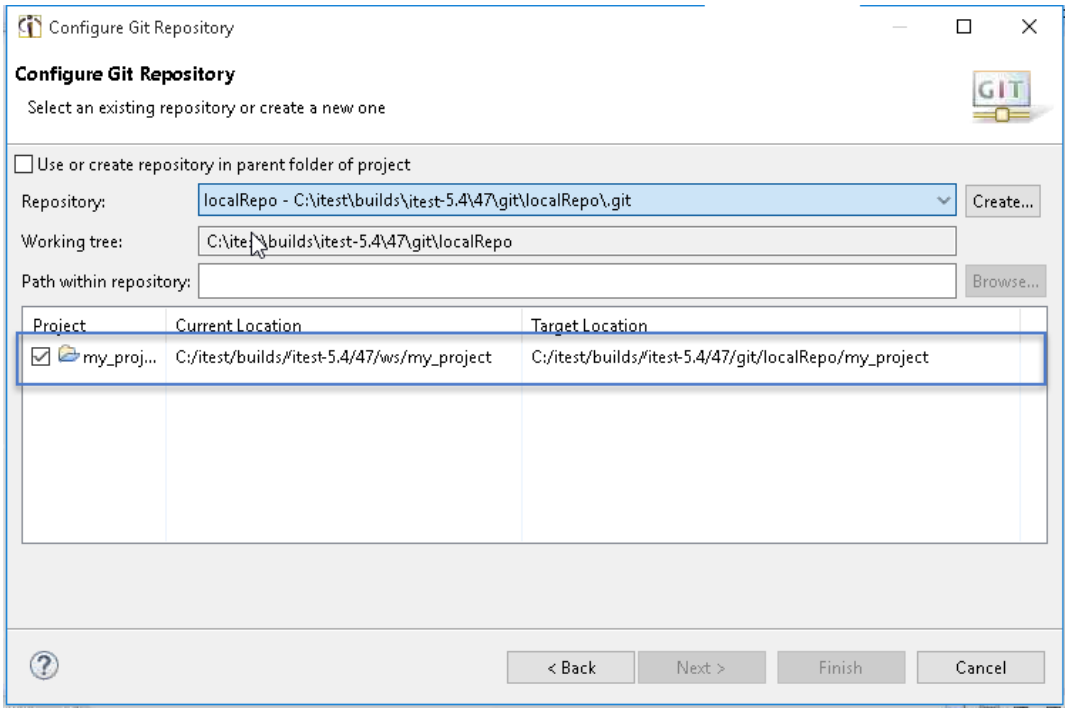
Step 1 Add your project folder to Git (local repository)

1 Go to **iTest Development Perspective** (select from the icons at the top right-hand side of iTest window).

2 From the iTest Explorer, select project to be added to Git. Right-click select **Team > Share Project**.

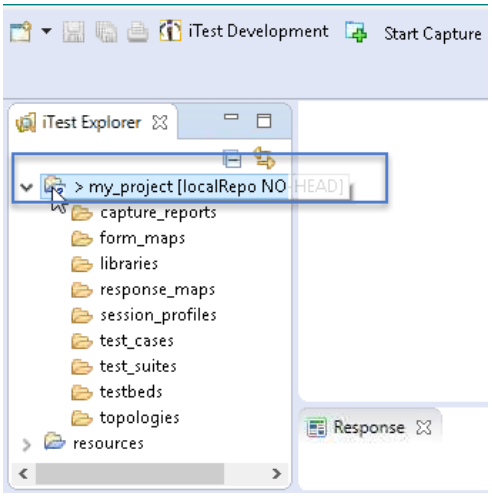


3 Select **Git** when the **Share Project** dialog opens. The **Configure Git Repository** dialog opens populated with the project folder you selected.



Select the appropriate **Repository** from the list and **Path within the repository** (if required)

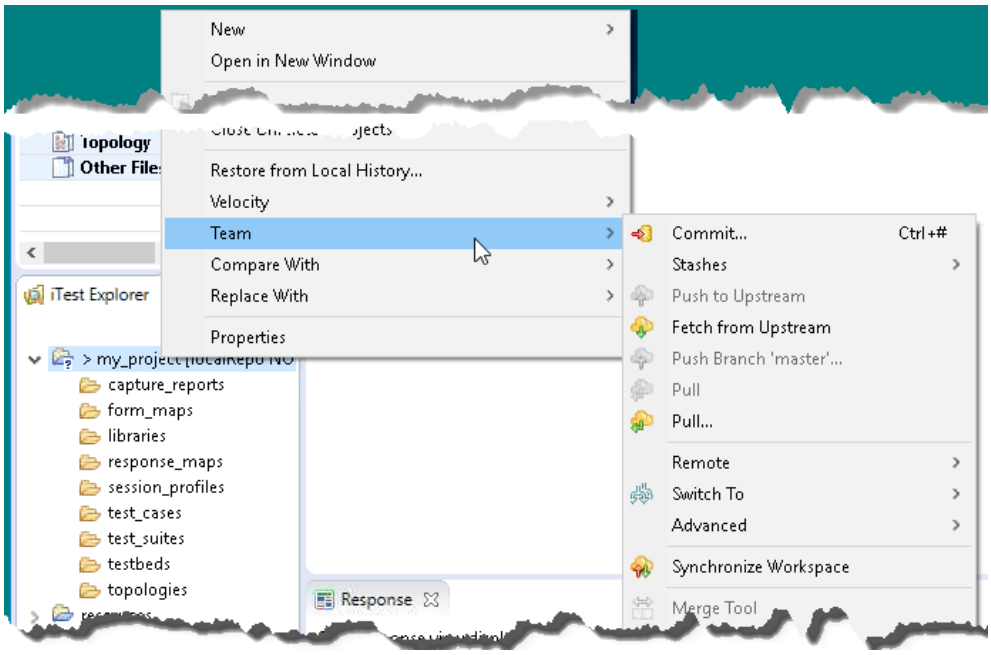
4 Click **Finish**. The selected project is added to **Git** and the iTest Explorer tree view displays as follows.



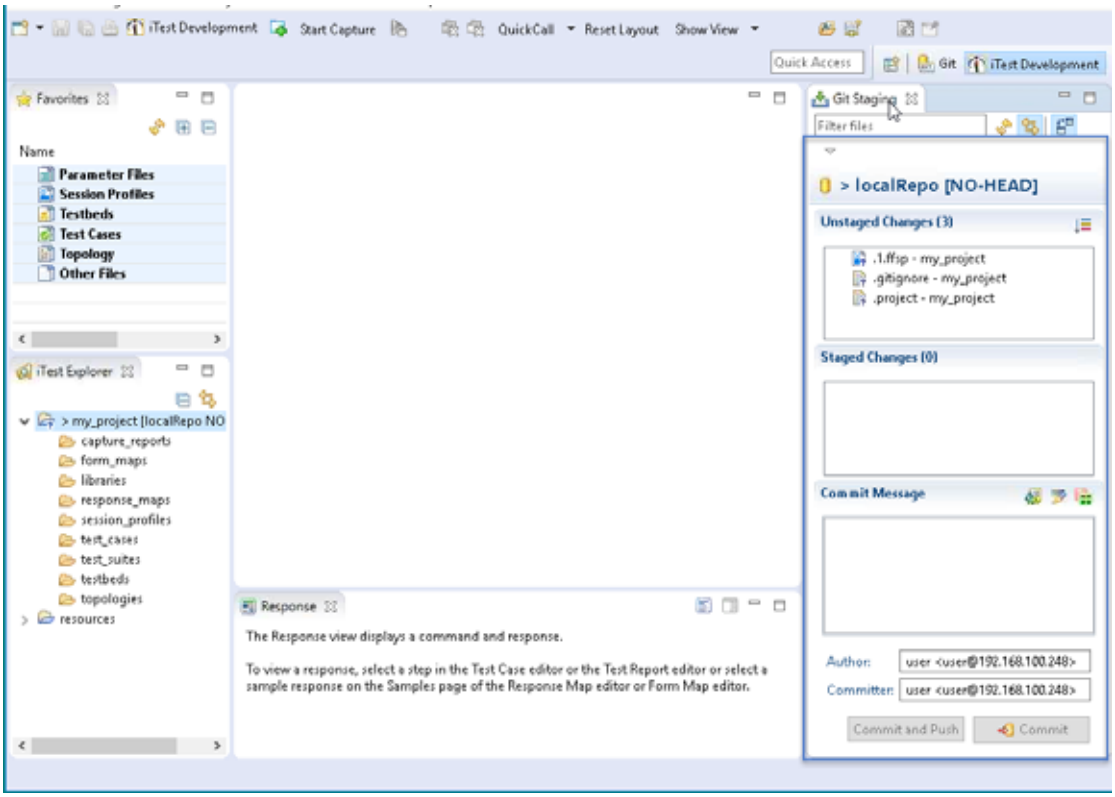
Step 2 Add your project folder to Git (remote repository)

After your project is in the Git local repository follow these steps to **push** it to the remote repository.

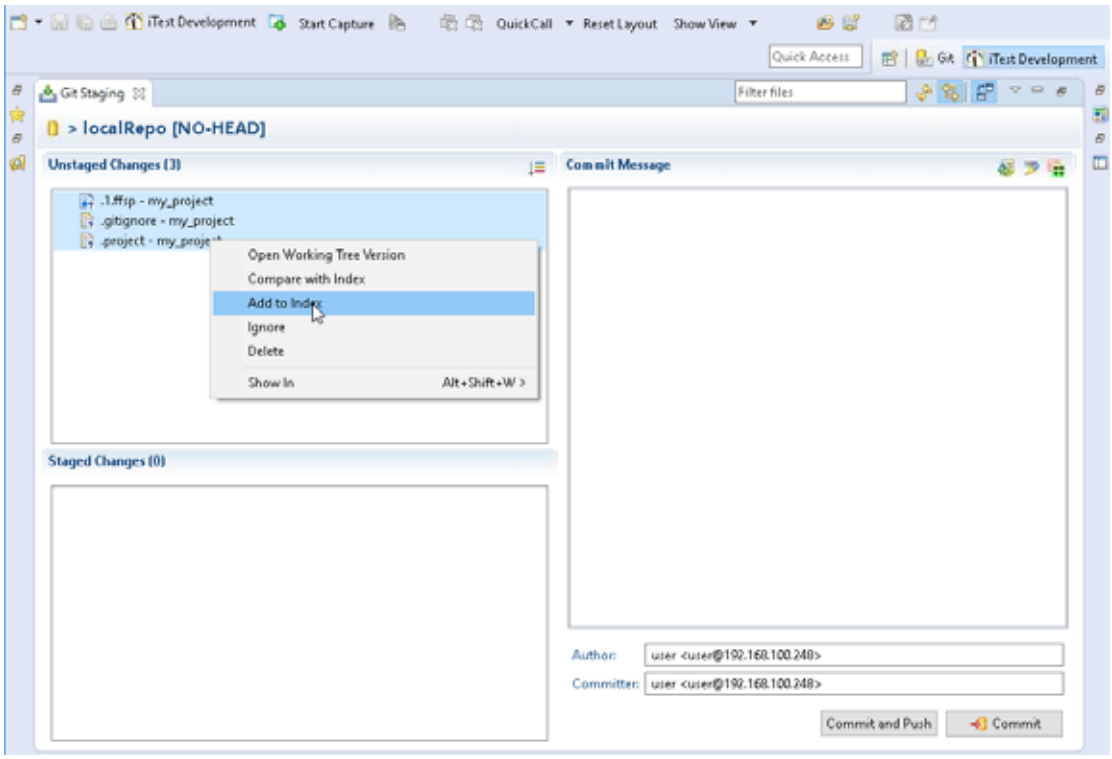
1 Select project added to **Git**. Right-click select **Team > Commit**.



2 The Git Staging tab displays with local project ready to be staged, committed, and pushed to the remote repository.

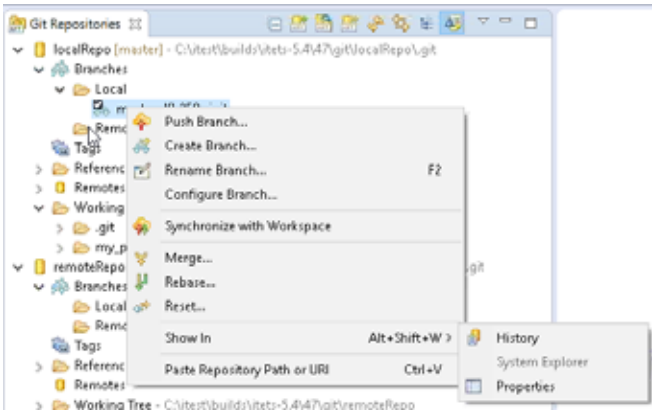


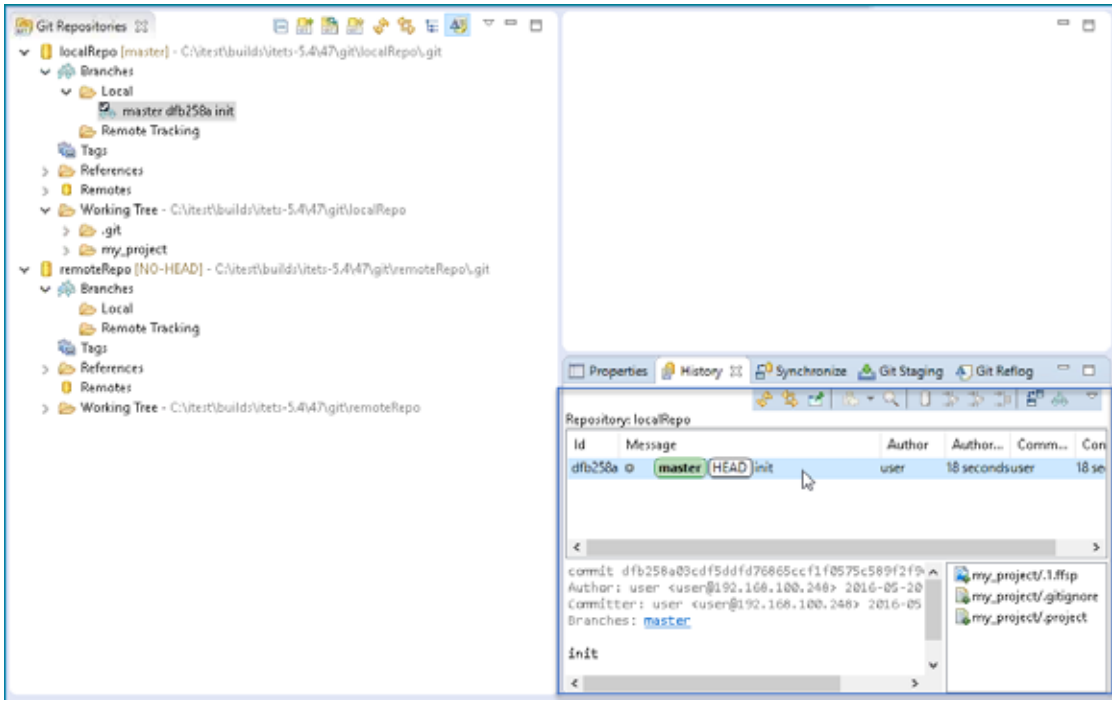
3 Expand the **Git Staging** tab, select and right-click the files in the **Unstaged Changes** section of the **Git Staging page.**



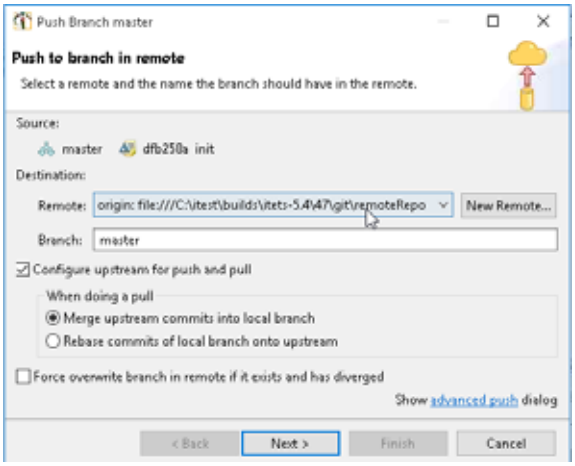
The project/files are staged as illustrated below. Add a commit message for clarify and verification purposes. Click **Commit**. .

After you click **Commit**, go to the **Git Repositories** page and notice that the project/files are ready to be added/pushed to the remote repository as illustrated below. You may view the commit history to verify the commit...

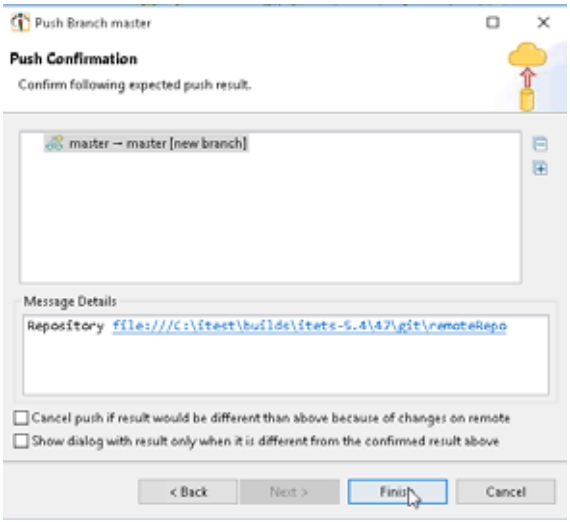




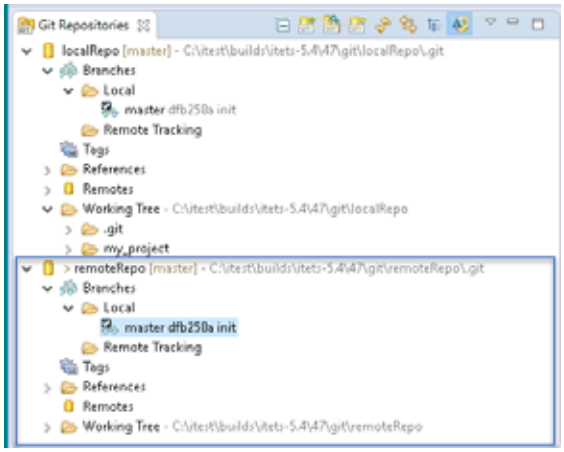
4 Right-click the local commit and then click **Push Branch**. Make sure that the project is pushed to the required repository.



Click **Finish** and a confirmation dialog displays as illustrated..



Go to the Git Repository page a verify as illustrated..



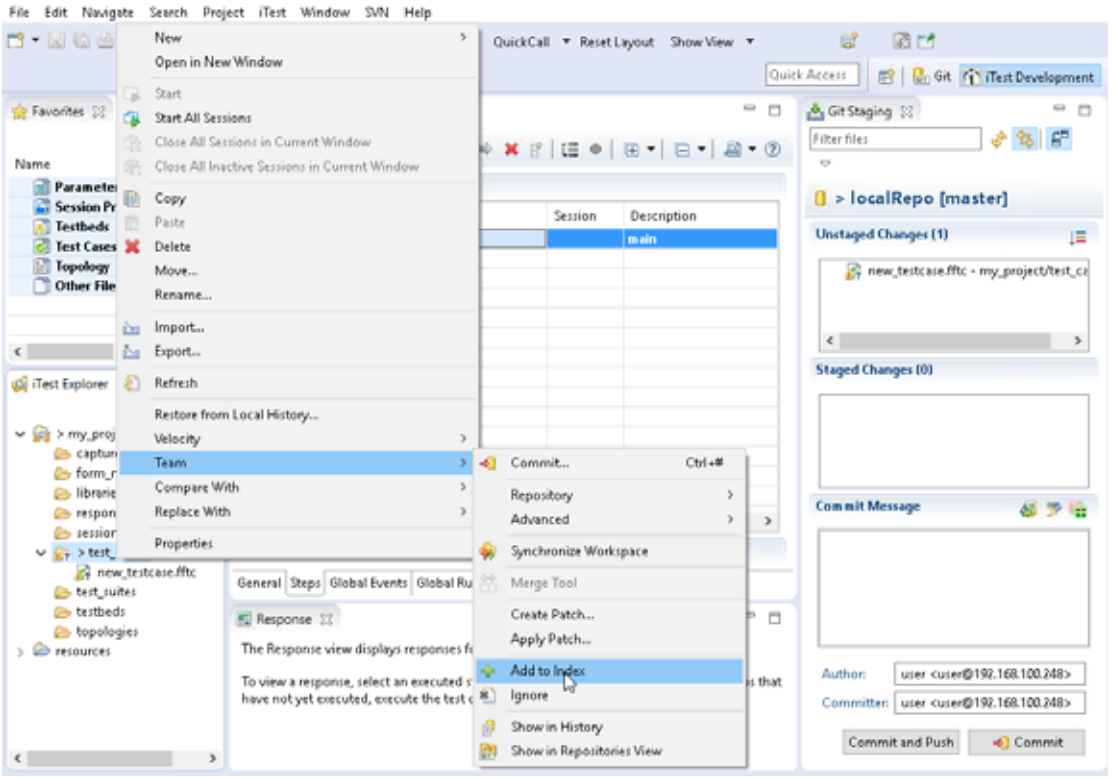
Adding New iTest Project Files (e.g.,Test Cases) to Git

This section provides instructs on adding new project files to Git. The examples illustrates adding a test case to Git.

Step 1 Adding a Test Case to Git

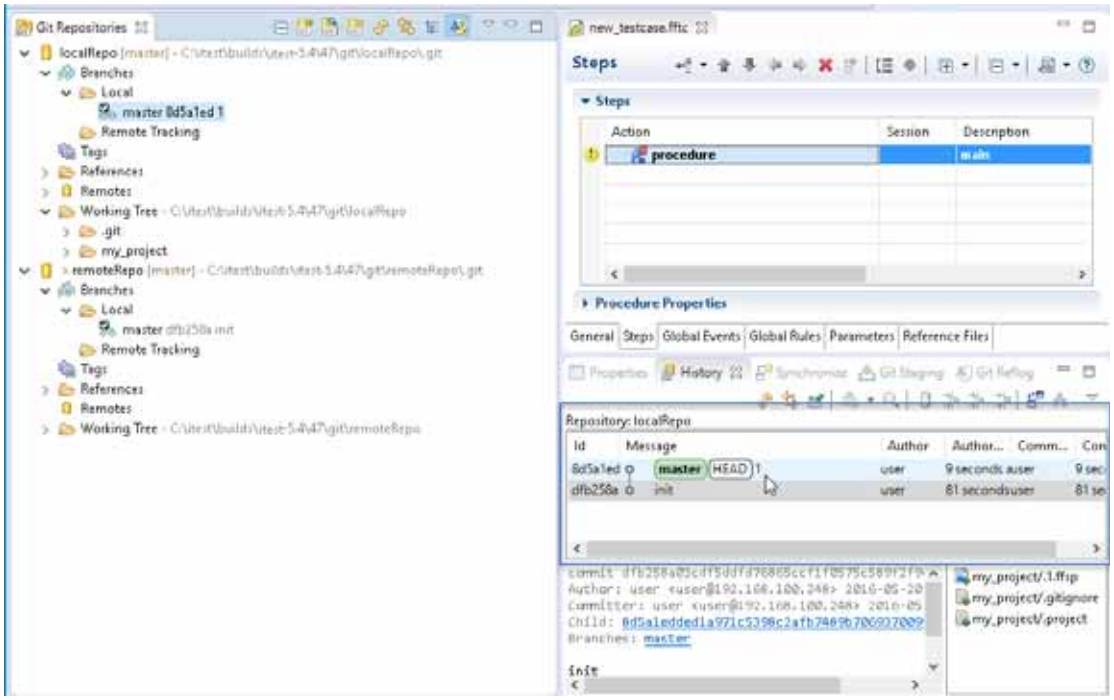
1 Create a a new test case (See Chapter 7, “Test Cases” on page 125) and save. The new test case appears in the **Unstaged Changes** section of the **Git Staging** page on the right. See illustration below.

2 Right-click the **test cases** folder in iTest Explorer view and then click **Team > Add to Index.**

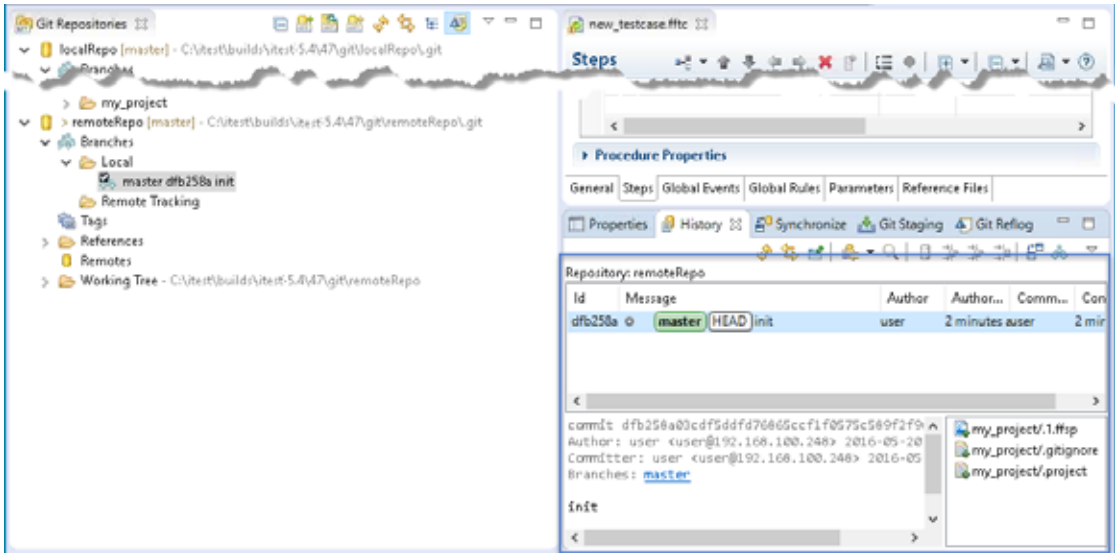


3 The Test case is staged when you click **Team > Add to Index.**

4 Add a commit message and click **Commit**. Verify the commit in local repository as illustrated.



Verify in the remote repository to confirm that the new change has not yet been pushed.



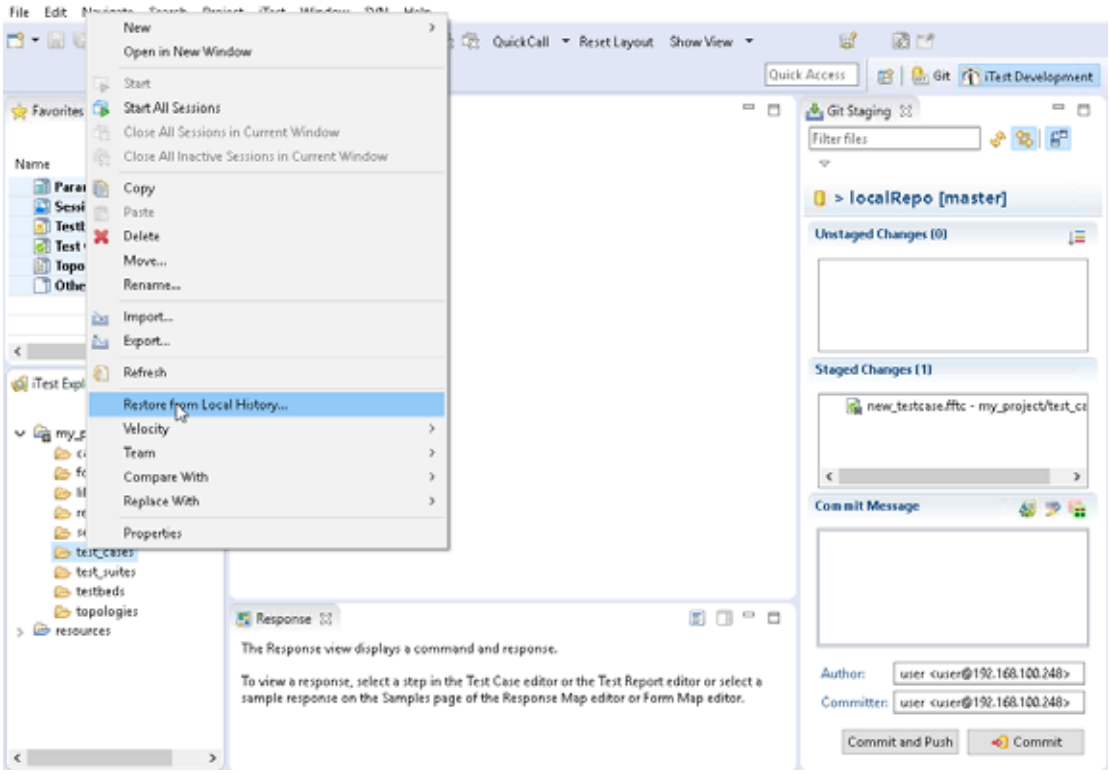
5 Right-click the local commit and then click **Push Branch**. Make sure that the project is pushed to the required repository. (Refer to the description and illustration in Step 4 page 1313).

Restore Deleted Files

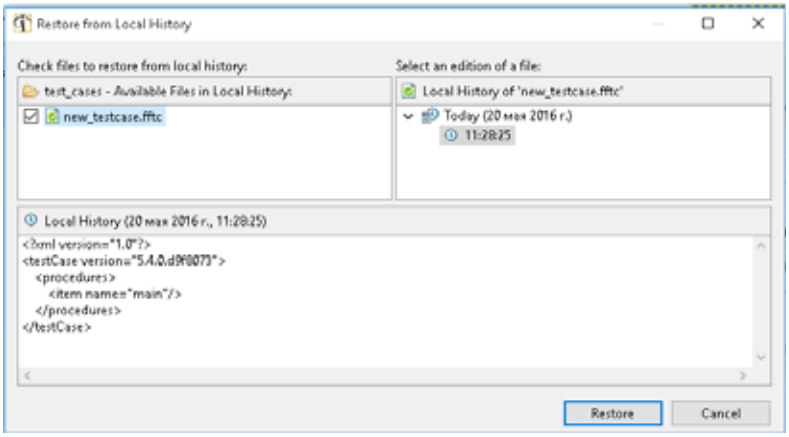
This section describes how to recover a file that was deleted in iTest from Git.

1 Go to iTest Development View, and delete a file you previously Staged.

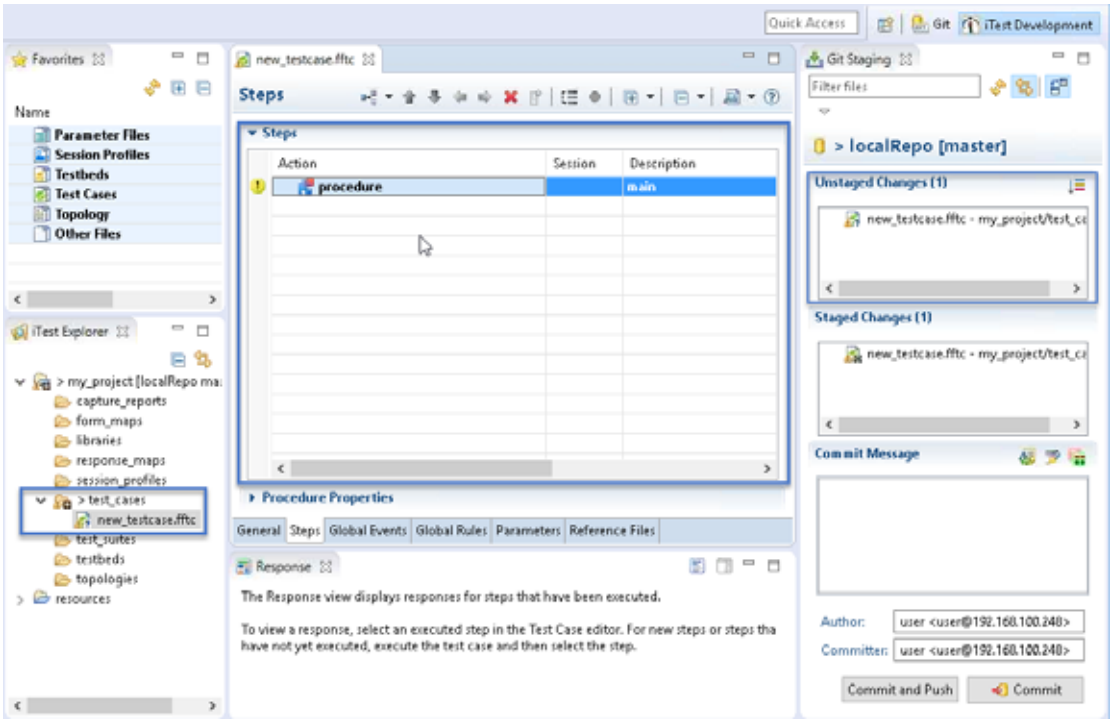
2 Right-click on the **test_cases** folder and then click **Recover from Local History.**



3 The Recover from **Restore from Local History** dialog opens. Select file to recover and click **Restore**.

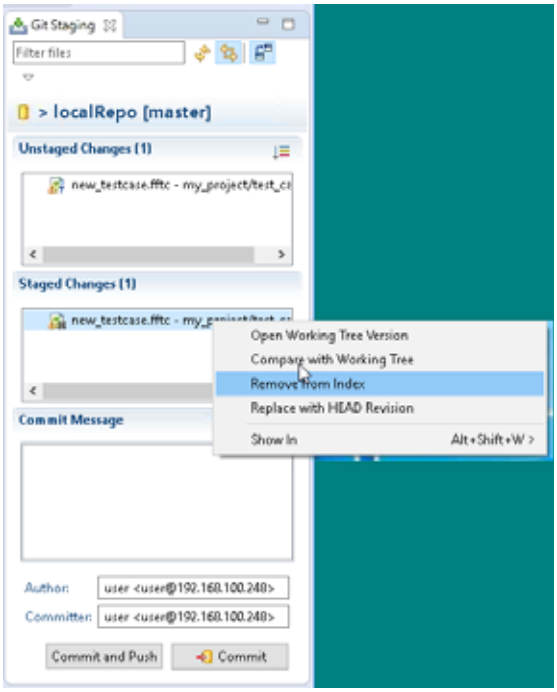


4 The restore file is populated in the iTest Explorer in Development view as illustrated below



You may delete the restored file from the Git Staging section, modify the restored file as required and commit again, if required.

5 Right-click file in the **Staged Changes** section of the **Git Staging** page to delete files that are not required.



You may also perform other tasks options available on the right-click menu in the Git Staging > Staged Changes section.

Setting preferences for Git

To view or edit preferences, click **Window > Preferences**. On the **Preferences** page, click **Team> Git.**

General information on setting and sharing preference settings appears in Chapter 41, “Configuring iTest Preferences”.

Team > Git

Default repository folder	Enter the folder path (where you to clone the master repository), for example, **C:\Users\user\git** **Note** In order for EGit to work in iTest, you should already have your main Git repository set up.

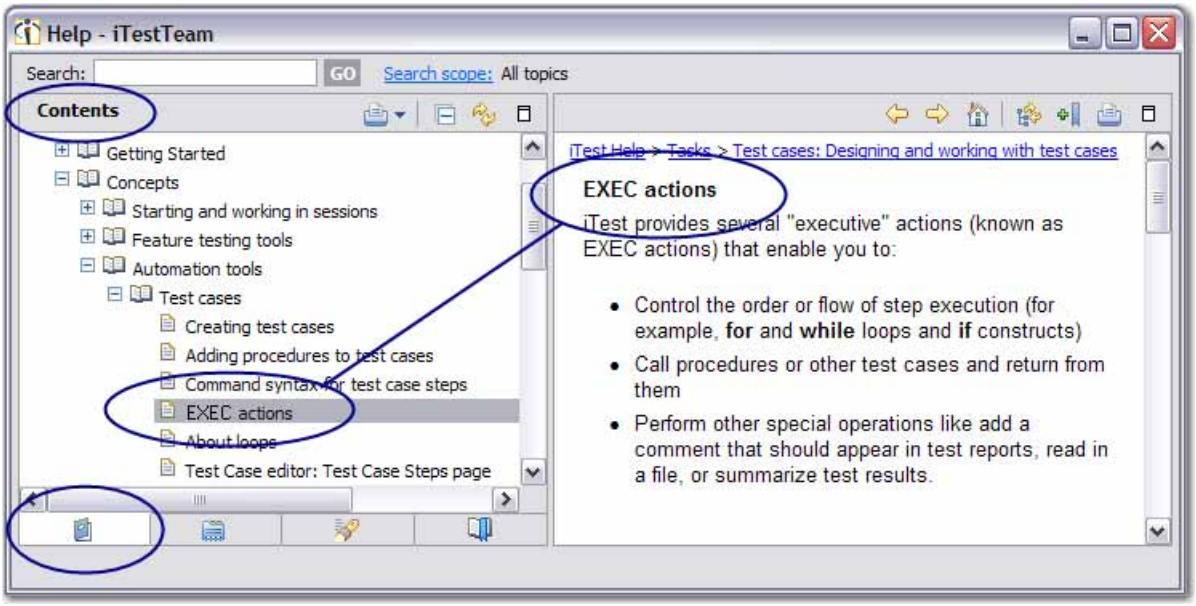
CHAPTER 65

Using iTest Help

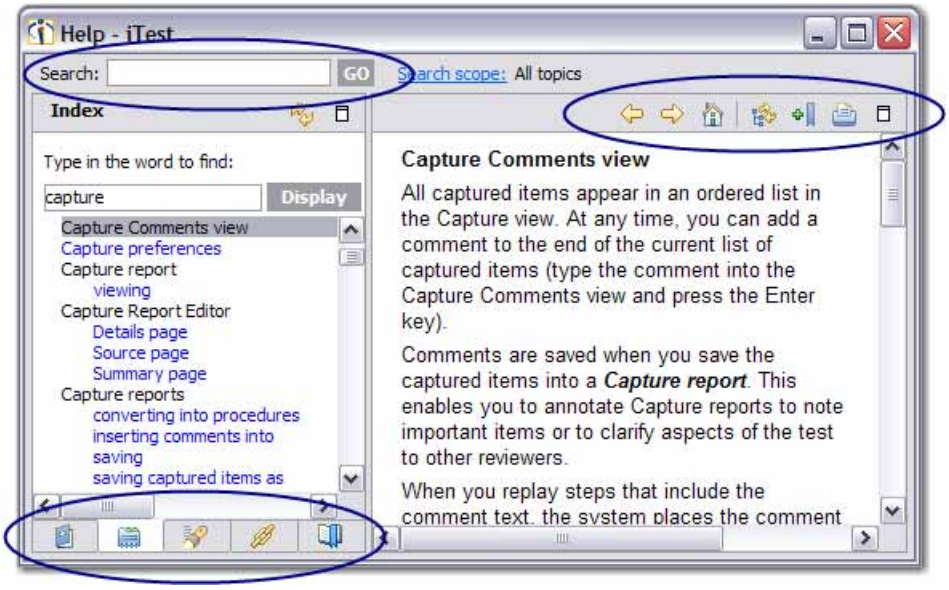
Using iTest Help

Browse topics in the **Contents** frame ( ) on the left.

Click a topic to view it.



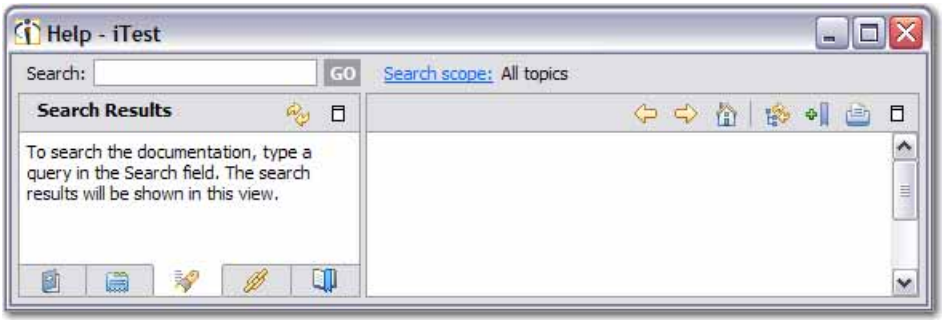
Use **Back** and **Forward**  to navigate through the history of viewed topics.



Click the **Index** tab ( ) to display the Index view. Type the word into the text box and then click Display. All related topics appear in the view.

After you run a search and find a topic, click either **Refresh / Show Current Topic** ( ) or the **Show in Table of Contents** ( ) to synchronize the navigation tree to the current topic location. You might also find it useful to synchronize after following in-topic links.

To locate topics on a particular subject, enter a query in the **Search** field. Click the **Search** tab ( ) to display the Search view. You can narrow the **scope** of your search by selecting the sections of interest before searching.



After you run a search and find a topic, click either **Refresh / Show Current Topic** ( ) or the **Show in Table of Contents** ( ) to synchronize the navigation tree to the current topic location. You might also find it useful to synchronize after following in-topic links.

Feedback on documentation and online help

At Spirent, we're happy only when you're happy, so we value your feedback.

If you have comments or suggestions about our online help or written material, please email us at **iTest_documentation@spirent.com** (this address is for documentation feedback only — be sure to send requests for technical support to **support@spirent.com**).

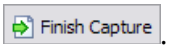
Please include the following information with your feedback:

- Product name and version number (use **Help > About** to view the information)
- For PDF or hardcopy manuals:
  - Title and revision number of the manual
  - Page number
- For online help:
  - Topic title
- Brief description of the issue: Poorly worded, does not reflect actual product operation, examples needed, and so on.
- Your suggested improvement, if appropriate.

Quick Tips

This topic presents quick overviews of several iTest tasks.

1. Start direct-to-test capture by clicking 

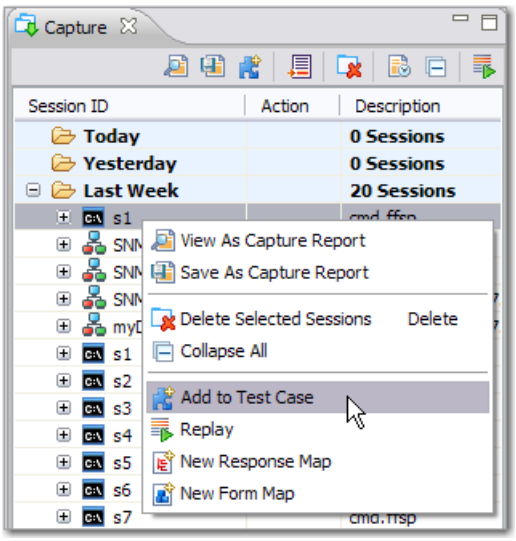
2. Open an existing session profile and start a session with a device (this opens a new Session window).
3. In the Session window, issue commands to the device.
4. Close the Session window.
5. Stop the capture process by clicking 

.
6. iTest opens a quick **Add To Test Case** wizard to let you save the steps immediately. You have the option of inserting steps into an existing test case or of saving whole sessions as steps in a new or existing test case.
7. The captured steps appear in the test case editor (creating the basic test case).
8. You can now replay the test case by clicking  (A new Execution window appears)
9. When execution ends, the Session window closes and a test report is displayed,
10. Now you can edit the test case as needed to add analysis for pass/fail, logic functions such as for loops, if statements, procedure calls, and so on.

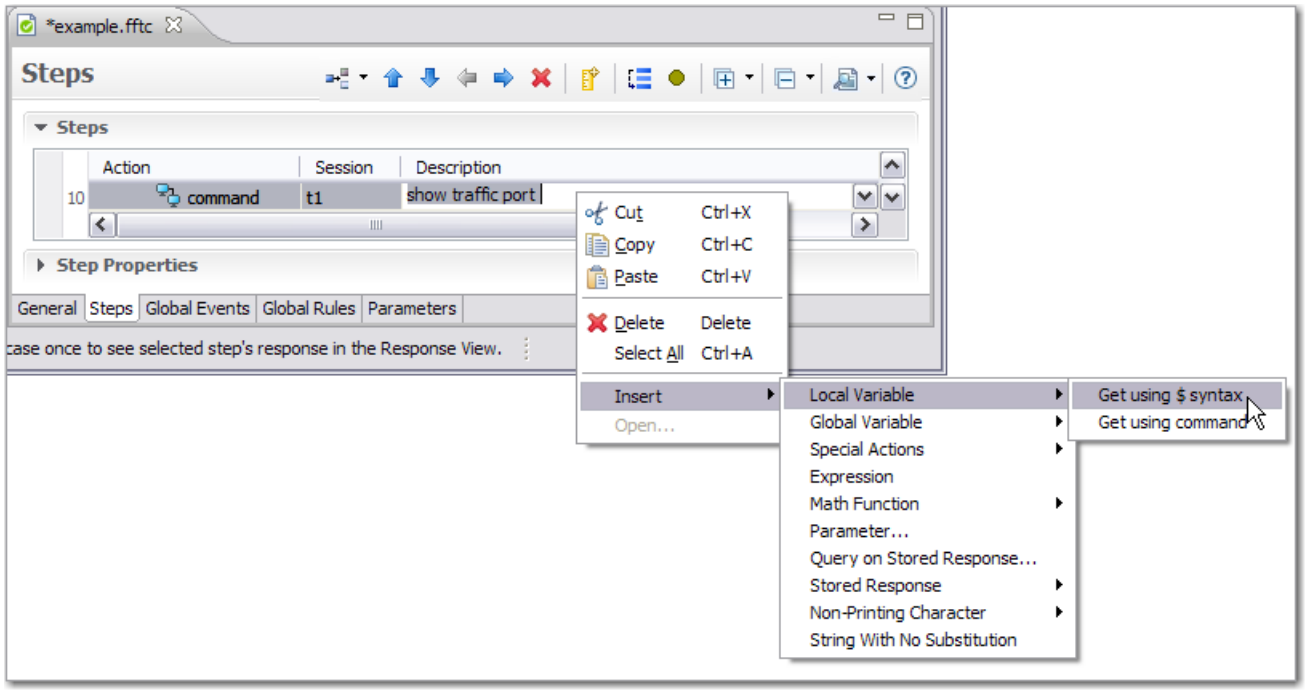
When in doubt, right-click

Most of iTest's editors and views include context (right-click) menus for common operations (most toolbar tools appear in the context menus).

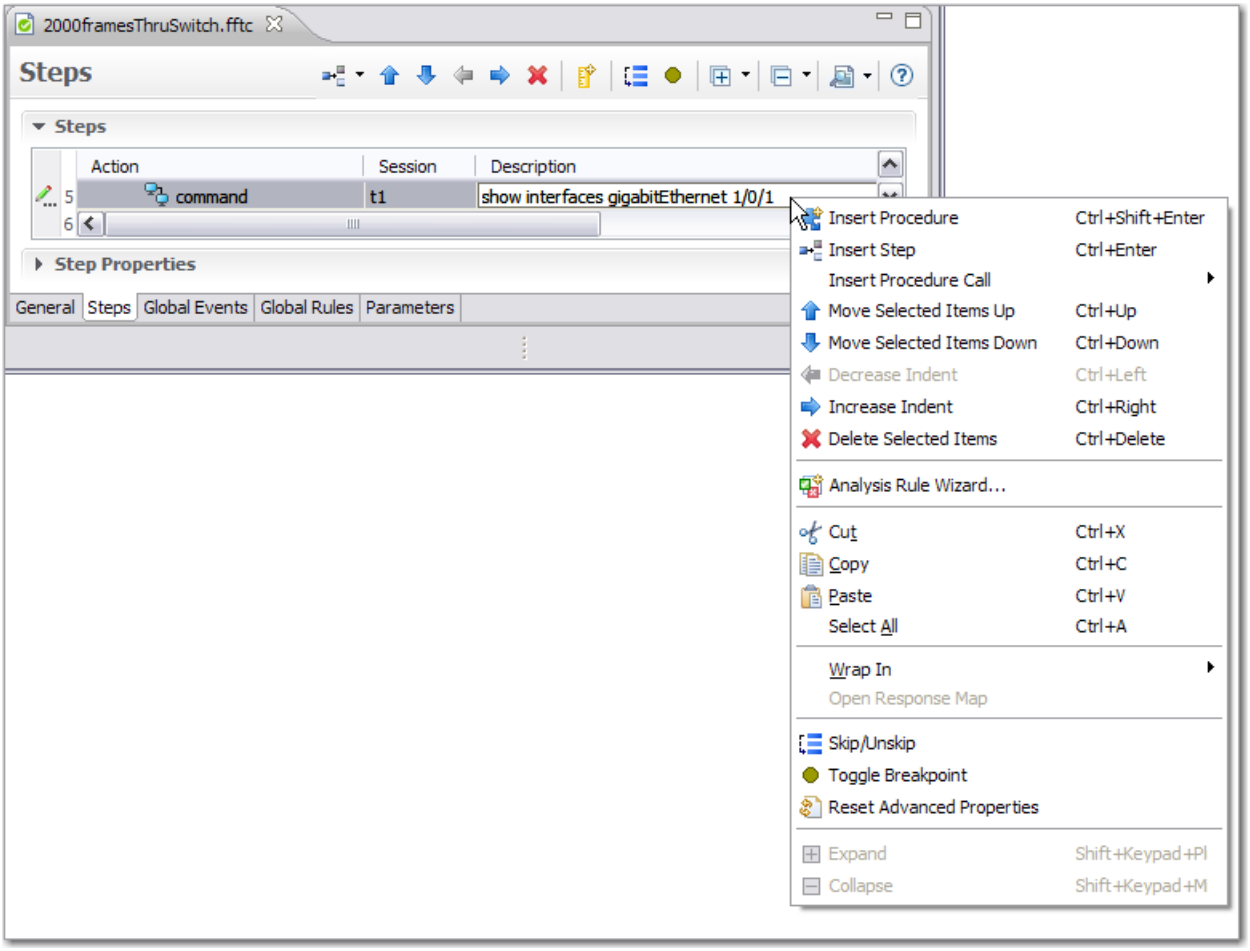
- For example, here's the context menu in the Capture view:



- In the Test Case editor, there are two menus:
- To insert an item (for example, a variable), select the item to replace or place the cursor at the appropriate location and right-click:



- To modify a step (skip it, wrap it inside an if statement, a loop, or a comment, apply an analysis rule, and so on), select the step and right-click a cell:



Getting Technical Support

Spirent is dedicated to providing the highest level of support for our customers. If you need assistance, please contact us. See Chapter 68, “How to Contact Us” for contact information.

If you have comments or suggestions on user documentation, please send a message to iTest_documentation@spirent.com

Getting to information

This online *User Guide* should help you to get started using iTest and will answer many day-to-day questions you may have. Here are several other sources of information:

- **Customer Training** at your site or ours — See your Account Manager or Application Engineer. You can download the training material from the Customer Support Portal (described in a moment).
- Contact Spirent to request a DVD of our video-based training.
- **Technical Notes** — Short papers that solve a particular testing problem that you face. See your Account Manager or Application Engineer.

- **On-line help** — Press **F1**, click , or click **Help > Help Contents** at any time to view online help.
- **ToolTips** — Position the cursor over any toolbar button and pause to view a brief description. For more detailed information, look up the item's name in the online help.
- **Sample code** — Spirent provides sample documents that you can use to guide you in creating your test cases.
- **Knowledge Base** on the Customer Support Portal: Technical articles, tips, sample code, documentation updates, notification of product updates, and other important technical support information.

Searching help

The help system includes a search engine that can run simple or complex queries on the documentation to help you find the information you are looking for.

To search help:

1. From the main menu, select **Help > Search**
2. Type in the word or phrase for which you want to search
3. Click **Go** or press Enter. The list of results will be displayed below
4. To view the content of a topic in the list of results, click it

Alternatively, you can search from the Help window using the **Search** field at the top of the window.

Refining the search results in the help view

If the search yields too many results, the information you are looking for may not appear in the top 10 or 15 results. You can then refine the search to reduce the number of results.

To refine a search:

1. Click the **Search Scope** link. to expand search scope section
2. Click the **Advanced Settings** link. The Search Scope preference dialog will open
3. Select **Local Help** from the list
4. Click **Search only the following topics** to narrow down the search scope
5. In the working set content tree, select the topics to which you want to narrow the search
6. Click **OK** to activate the changes and return to search page in the Help view
7. Click **Go** again. The new list of results will appear

Follow the following search expression rules for searching local help content:

- Unless otherwise stated, there is an implied AND between all search terms. In other words, topics that contain all the search terms will be returned. For example:
- Java project
- returns topics that contain the word *Java* and the word *project*, but does not return topics that contain only one of these words.

- Use OR before optional terms . For example:
- applet OR application
- returns topics that contain the word *applet* or the word *application* (or both).
- Use NOT before terms you want to exclude from search results. For example:

  ```
  servlet NOT ejb
  ```

  returns topics that contain the word *servlet* and do not contain the word *ejb*.

  Note NOT works only as a binary operator (for example, “NOT servlet” is not a valid expression).

- Use ? for a single-character wildcard and * for a multi-character wildcard. For example:

  ```
  par?
  ```

  returns topics that contain *part* or *park*, but not *participate*. On the other hand:

  ```
  par*
  ```

  returns topics that contain *part*, *park*, *participate*, *pardon*, and so on. **Note:** The search engine does not accept terms with a wild card at first character position.
- Use double quotation marks around terms you want treated as a phrase. For example:

  ```
  "creating projects"
  ```

  returns topics that contain the entire phrase *creating projects*, and not *creating* or *project* on its own.
- Punctuation acts as term delimiters. For example:

  ```
  plugin.xml
  ```

  returns hits on topics that contain *plugin.xml*, *plugin*, and *xml*, which is likely broader than you want. If you want to find just those topics containing *plugin.xml*, use double quotes, as in:

  ```
  "plugin.xml"
  ```
- The search engine ignores character case. For example:

  ```
  Workbench
  ```

  returns topics that contain 'workbench', 'Workbench', 'WorkBench', and 'WORKBENCH'.
- The following stop words are common English words which will be ignored (not searched for) if they appear in the search expression: a, and, are, as, at, be, but, by, in, into, is, it, no, not, of, on, or, s, such, t, that, the, their, then, there, these, they, to, was, will, with.
- The search engine does “fuzzy” searches and word stemming. If you enter *create*, it will return hits on topics that contain *creates*, *creating*, *creator*, and so on. To prevent search engine from stemming terms, enclose them in double quotes.

Extending the search scope

If you cannot locate information in the local help, you can extend search scope to remote info-center or search engines.

To enable search engines:

1. Click the **Search Scope** link. to expand search scope section. The list of search engine is displayed.
2. Select the ones that contain information you are looking for.

In addition to search engines provided, you may define additional search engines.

To define a new search engine:

1. Click the **Search Scope** link. to expand search scope section.
2. Click the **Advanced Settings** link. Search Scope preference dialog opens.
3. Click **New**.
4. Select the search engine type.
5. Click **OK**.
6. Provide a name and a description
7. Select engine specific settings and scope below. For the remote search engines, accessed using URI, fill in a full URI to query the engine. Use `{expression}` in the place of search expression.

Defining multiple search scopes

By default, changing search scope modifies the search scope named “default”. You can define multiple search scope. They will be saved, allowing to quickly change search scope to one of them.

To define a new search scope:

1. Click the current search scope name, beside the **Search Scope** link. Search Scope Sets dialog appears.
2. Click **New**.
3. Type a name, and confirm.
4. Select the newly created search scope.
5. Click **OK**. The new search scope becomes current.

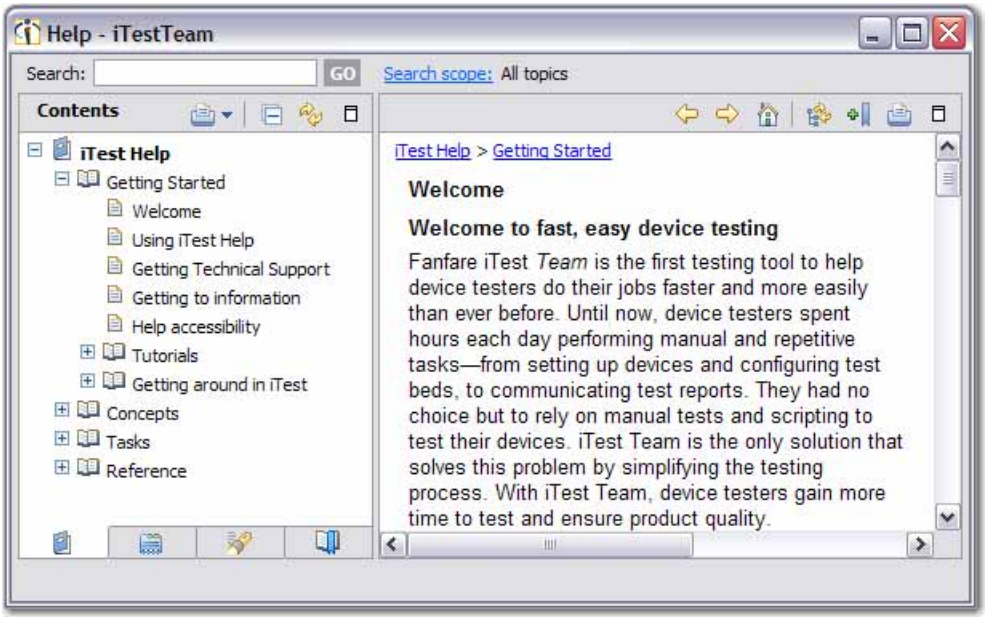
Changes to the search scope affect current search scope.

Navigating help topics

You can browse help topics using the Help window or Help view. Which one you use is simply a matter of preference; the view is well integrated with the workbench and is good for quick help lookups. The Help window, on the other hand, has more real estate and may be preferable for longer reading.

Using the Help window

The Help window is a window separate from the workbench used exclusively for browsing, searching, and printing help. To open the window, select **Help > Help Contents** from the menu. This opens the help window with the Contents view visible, which shows the table of contents.



To navigate the Help window:

1. Find the topic you want to read in the table of contents by clicking to expand the subtopics.
2. Most topics provide a list of links to related topics at the bottom. Follow these links to learn more.
3. Use **Go Back** and **Go Forwards** to navigate back and forth. These behave the same way as in Web browsers.

The different parts (views) of the Help window can be maximized to take up the entire window.

To maximize a frame, click **Maximize** in the toolbar, or double click the view's title header.

To return the view to its original size, click **Restore** or double click the title header again.

To print a topic from the Help window:

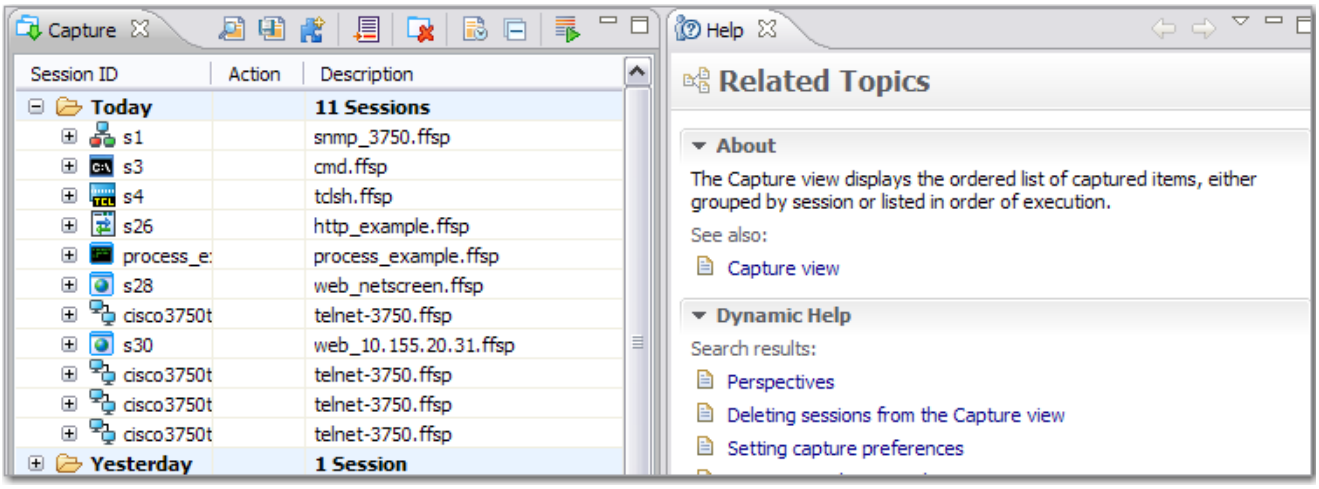
a. Select the topic in the navigation frame.

Click **Print** in the Help toolbar.

b. Select the desired printer settings, and click **Print**.

Using the Help view

The Help view provides the same features as the Help window (with the exception of printing), but in the same window as the workbench instead of a separate one. To open the Help view, select **Help > Dynamic Help** from the main menu. This will open the Related topics page.



To navigate the Help view:

1. Switch to the **All topics** page using the link at the bottom.
2. Find the topic you want to read in the table of contents by clicking to expand the subtopics.
3. Most topics provide a list of links to related topics at the bottom. Follow these links to learn more.
4. Use **Go Back** and **Go Forward** to navigate back and forth. These behave the same way as in Web browsers.

Capabilities

To show documentation about capabilities that are disabled in the application, click **Show All Topics**. When you choose to show all topics in the table of contents, the headings for documentation about any disabled activities are shown in the table of contents and also appear in search results.

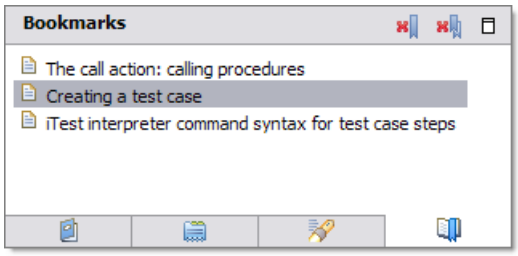
Show in table of contents

While reading a topic, you can find out where the topic is located in the table of contents by clicking **Show in table of contents** in the toolbar. This will display the table of contents with the topic highlighted.

Bookmarking a page

You can “bookmark” any help page that you find useful.

- To view the list of bookmarks, click the **Bookmarks** tab.



- To add a bookmark: While viewing a page, click **Bookmark Page** in the main toolbar. The bookmark is added to the end of the list.
- To delete bookmarks, click **Delete Bookmark** or **Delete All Bookmarks**.

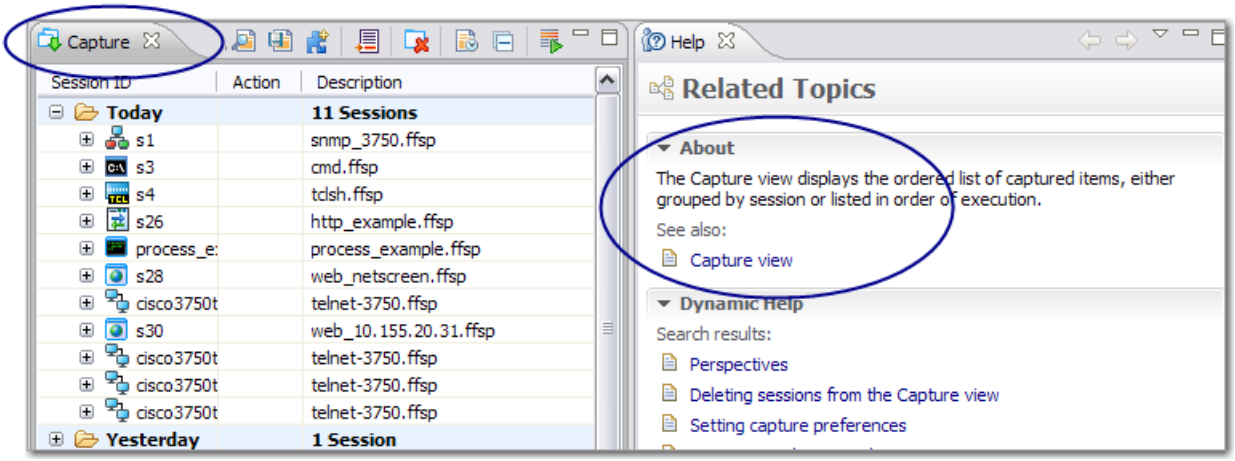
Accessing context-sensitive help

To access the context-sensitive help for a given widget:

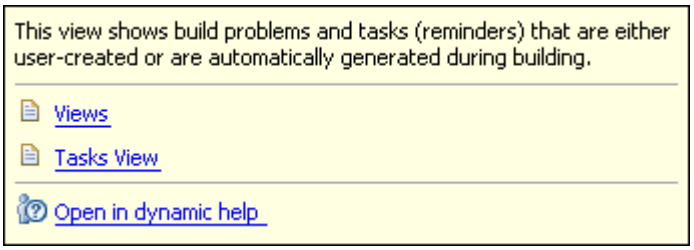
1 Select the widget by putting focus on it (see the table below).

2 Press the **F1** key (Shift+F1 on Linux, **Help** on the Mac). In dialog boxes, click .

This displays a description of the selected widget, and usually, a list of links to related information, in the help view. If you want more information than what is shown in the description, click a link to one of the related topic, or one of the search results appearing in the dynamic help section in the help view.



Depending on help preference settings, requesting context-sensitive help may display context help in an *infopop* instead of help view. You can dismiss the infopop by clicking outside it, or by pressing `Esc`.



If you want more information than what is shown in the infopop, click a link to the related information. This opens the Help browser to the selected topic and closes the infopop. The list of related links stays in the left hand frame, so you do not have to press F1 again if you are interested in other topics related to the same widget.

Note To view tooltip help for toolbar buttons, hold the cursor over the button.

Help display settings

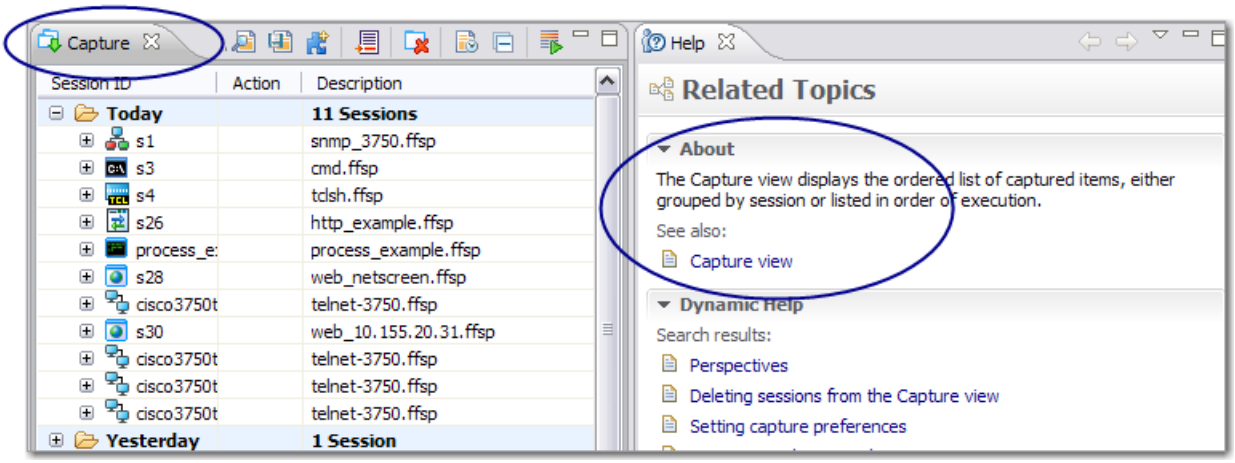
External browser

The Help system can be accessed in Help view, or separate Help window. The window can be the default window with an embedded browser, or a full external browser of your choice.

If the embedded browser is supported on your system, help will use it by default to display help. If you prefer to always use a full external browser, you may select this behavior in the **Help preference page**.

Context-sensitive help

By default, context-sensitive help for editors and views is displayed in the Help view. Context-sensitive help for dialog boxes is displayed in the dialog's tray, which is similar to the Help view.



Displaying topics

A topic selected in the Help view can be displayed in place, in the Help view, or in the editor area. You can set this preference in the Help preference page. On some platforms, with no embedded browser, the topics will always be displayed in an external browser.

Help accessibility

The help browser uses your operating system's settings for the font colors, styles, and sizes. Users with visual impairments may wish to change some of these settings to increase the readability of the documentation.

In addition, on Windows platforms using Microsoft Internet Explorer, the help browser uses a component of Internet Explorer to display documentation, so changes you make to its display settings also affect the help display. To change the help browser's font and color settings:

1. Open Microsoft Internet Explorer.
2. Select **Tools > Internet Options**.
3. On the **General** page, click the **Colors, Fonts**, or **Accessibility** button.
4. Set the formatting options you desire.
5. Optionally, you can specify a cascading style sheet (CSS) to apply to the content.
6. Click **OK** and exit Internet Explorer.
7. Restart the Workbench. Open the Help perspective and browse the documentation to see the changes.

Note The help system also uses the Icon font setting on the Display Properties **Appearance** tab.

For more information about creating a CSS, consult a CSS reference. The W3 Consortium (http://www.w3.org) has an extensive collection of information about CSS and links to valuable resources.

CHAPTER 66

Wizards and Dialog boxes

Command dialog box

Use the **Command** dialog box to enter multi-line commands.

Alternatively, you can use the following settings to specify that, during execution, the step should use the value of a parameter as the command text.

Use the value of a parameter as the command for this step	When you check the box: • The **Command** property on the **Parameter** properties page displays the following text: “Command extracted from parameter <param_name>” • In the steps grid, the **Description** cell for the step displays the same text. • You must specify the **Parameter** property. **Note** If the parameter is masked, then, for the executed step, the command, the text response, and the structured data of the response are all cleared and are not displayed in any view or report. Default: unchecked
Parameter	If you check **Use the value of a parameter**, then type the name (or query) of the parameter to use.

iTest Import wizard

Use the import wizard to import a variety of document types and iTest projects.

General

Archive File	Import an archive file that was generated from iTest (using Export > Archive File).
Existing Projects into Workspace	Import projects from another workspace into the current workspace.
File System	Import files from your computer's/network's file system.
Preferences	Import a file that holds the preference settings that you configured on the **Preferences** page. See “Exporting and importing iTest preference settings” on page 861.

CVS

Projects from CVS	Import a project that had previously exported for inclusion in the CVS version control system. See the Eclipse help on CVS and Working with projects.

FanfareSVT

Device Definition Importer	Import FanfareSVT Device Definition documents and convert them into iTest response maps.

iTest

External iTest Session	Import iTest sessions from another instance of iTest (perhaps sessions that were created by coworkers).	
Test Report	Import test reports (perhaps reports that were generated by coworkers). See “Sharing test reports as files” on page 449	
Selenium IDE test case (HTML)	Import Selenium test cases. These test cases are intended for simple record-and-playback of interactions with the browser. The following lists the Selenium IDE commands recognized in iTest.	
	open type(AndWait) answerOnNextPrompt addSelection(AndWait) removeSelection(AndWait) click(AndWait) chooseCancelOnNextConfirmation	chooseOkOnNextConfirmation(AndWait) runScript(AndWait) select(AndWait) selectWindow submit(AndWait) goBack(AndWait) fireEventAndWait waitForPopUp mouseOver(AndWait)

Team

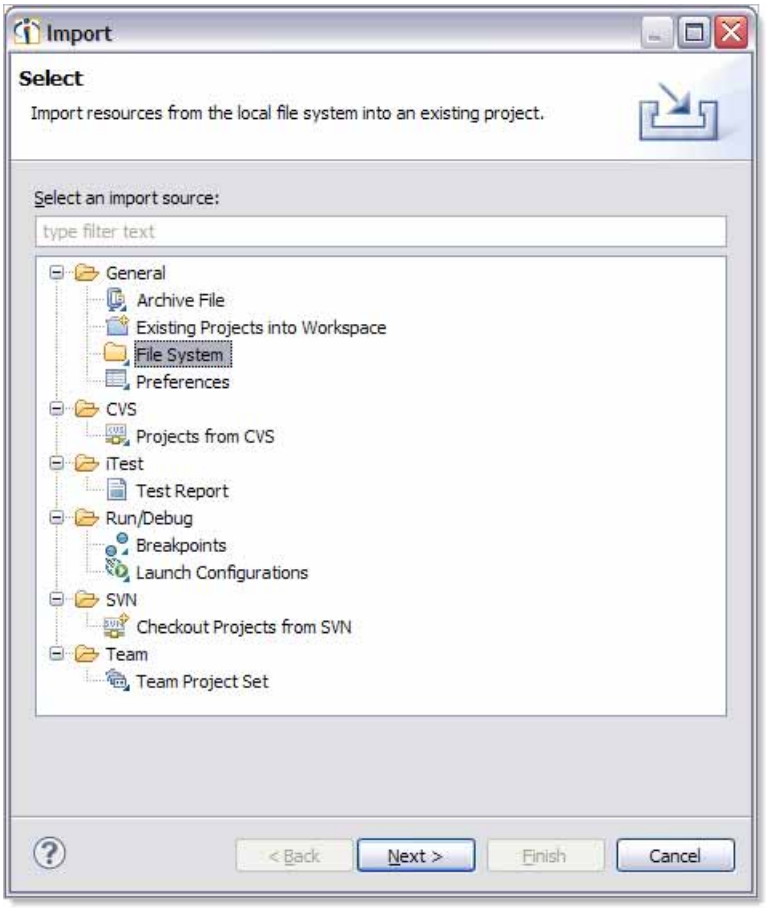
Team Project Set	Import projects that had previously been exported. See the Eclipse help on CVS.

Other

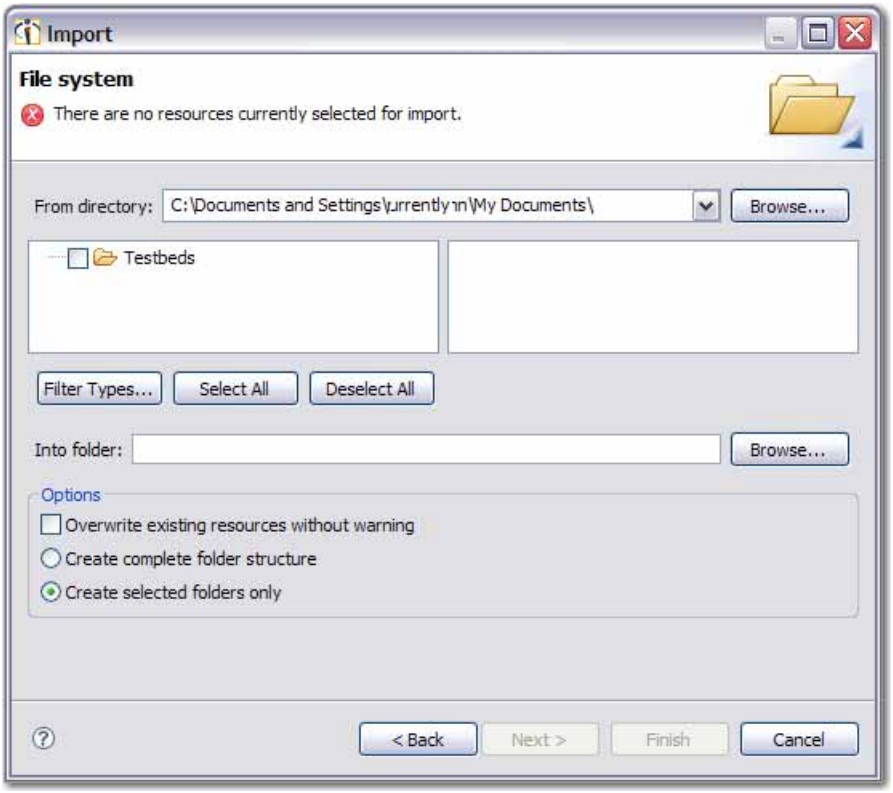
Checkout Projects from SVN	Checkout projects that are version controlled using Subversion. See the Eclipse help on Subversion.

Example: Importing files from the File system

1 Click **File > Import** and select the type of document to import (**File system** in this example).

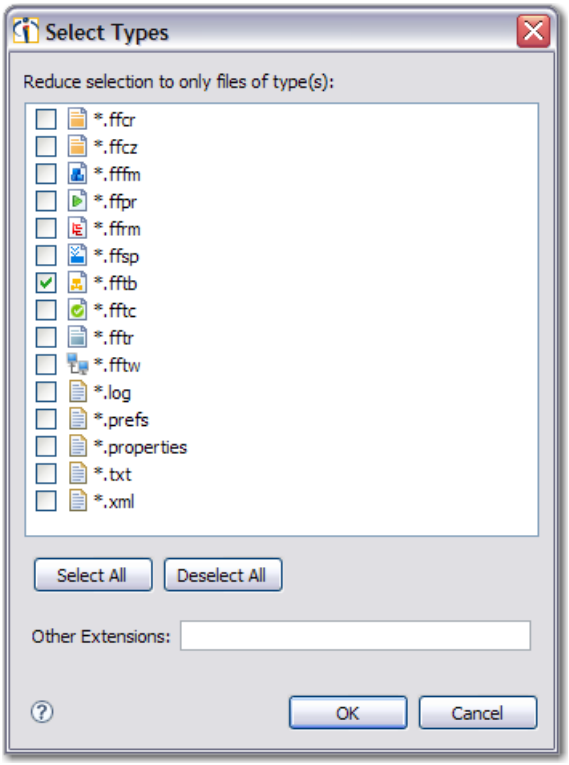


2 Populate the **From directory** box by browsing to the directory that holds the files to be imported

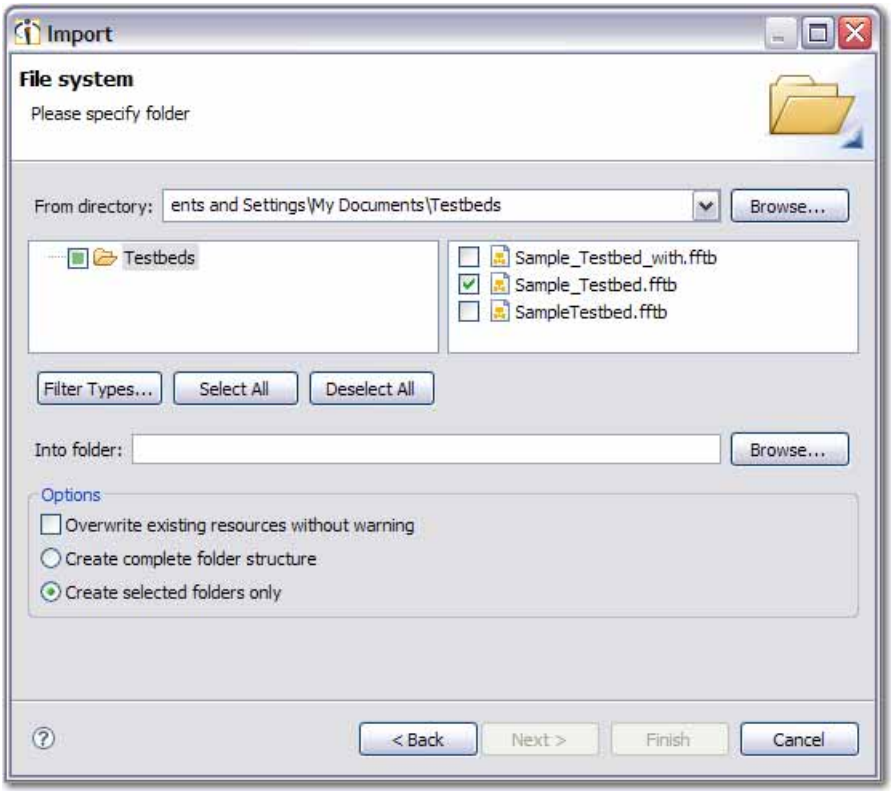


3 Now we select the filter that selects the type of file that we are interested in. Click **Filter Types**.

4 On the **Select Types** dialog box, we select ***.fftb** because that is the file extension for iTest testbed files. Click **OK**.



5 Now, on the **File system** page, click the folder in the left box and all of the files in the folder that pass the filter appear in the right box.



6 Select the files to import.

7 Supply the destination folder in the **Into folder** box by typing or browsing. In this example, we select the **Testbeds** folder in the project named **project**. (Notice that you can import the documents only into the current workspace.)

8 Click **Finish**. The files are imported and are now available to iTest.

New File dialog box

You can create simple text files using iTest. Use one of the following methods:

- In the iTest Explorer, right-click the intended parent folder and select **New >File**.
- In the main menu, click **File > New > Other > iTest> File**.
- In the main toolbar, click **New Document** and select **Other > iTest> File**.

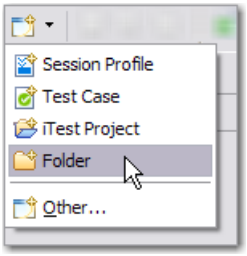
New Folder dialog box

To add a folder to the iTest Explorer:

1 Open the New Folder dialog box using one of the following methods:

- In the iTest Explorer, right-click the parent folder and select **New > Folder**.
- In the iTest Explorer, select the parent folder. In the main menu, click **File > New > Folder**.

- In the iTest Explorer, select the parent folder. In the main toolbar, click **New Document** and select **Folder**.



2 On the New Folder dialog box, select the parent folder in the **Container** field.

3 Type a name for the new folder in the **Folder Name** text box.

4 Click **Finish**.

New Session Profile dialog box

When creating a new session profile, you specify its location and provide a filename for the session profile in this dialog box.

Container	Path to the new session profile file. Default: **/my_project/session_profiles**
File name	Name of the session profile document that you are creating (**.ffsp** filename extension)

iTest saves the session profile document and then opens the new profile on the **Start** page of the Session Profile editor to enable you to configure additional session properties (IP address, prompt definitions, and so on).

New Test Case wizard

In addition to creating test cases by saving captured sessions, you can create a test cases "from scratch" by opening a new blank test case document.

1 Create a new test case document using one of the following methods:

- In the iTest Explorer, right-click the intended parent folder and select **New > Test Case**.
- In the main menu, click **File > New > Test Case**.
- In the main toolbar, click **New Document** and select **Test Case**.

2 The **New Test Case** wizard starts. Specify the **Container** (the parent folder). When you click **Browse**, the explorer box displays all projects in the current workspace and selects a recommended folder. You can navigate to any project or folder.

3 Type a name for the new document in the **File name** text box.

4 If you select **Associate this test case with a test in Quality Center**, then the wizard presents additional pages where you configure the connection to the QC server and specify the QC test and test instance to associate with the test case. See "Creating a new iTest test case that will be associated with ALM test" on page 950.

5 Click **Finish**.

Save As dialog box

On this dialog box, you specify the directory path (folder) and name of the document to save.

1. Specify the **parent folder**. The explorer box displays all projects in the current workspace and selects a recommended folder. You can navigate to any project or folder.
2. Type a name for the new document in the **File name** text box.
3. Click **Finish**.

Selecting a session profile

The **Select Session Profile** dialog box enables you to specify which session profile to edit or to add to a device definition.

1. Select either **Workspace** (the document is in the current iTest workspace) or **File system** (the document is in an itar file somewhere in the file system).
2. Browse to the appropriate session profile.
3. Select the profile and then click **OK**.

Selecting a session profile or device

The **Select a Session Profile or Device** dialog box enables you to specify which session profile or device definition to use to launch a session.

For Test Case, the **open** step allows selecting device type and session profile, and disables application type.

- **Device:** The box displays each device definition associated with the current test case a topology, testbed, or global testbed specified for the test case). Select the device from the list and then click **OK**.

  The text in the **Description** cell will be a device URI of the format **device:device_name**. For example, **device:telnet_DUT6** means: **"**From the topology, testbed, or global testbed specified for the test case (on the Test Case editor **General** page), fetch a device definition named **telnet_DUT6**."

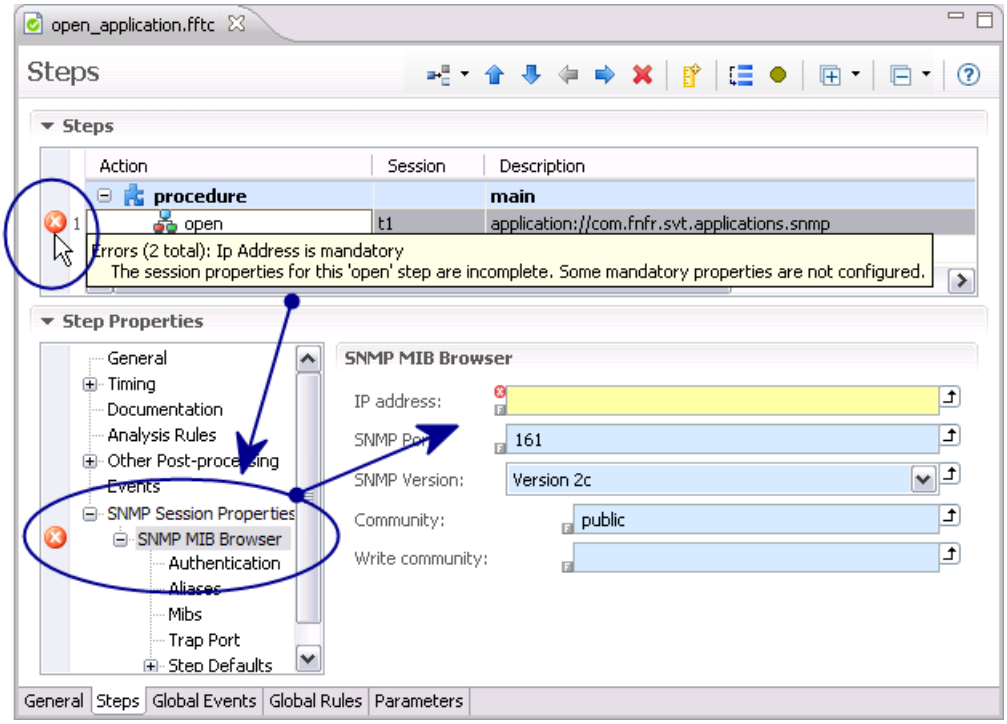
- **Session profile or reference session profile:** The box displays a tree of all projects in the current workspace. Navigate to the session profile and then click **OK**.

  The resulting text in the **Description** cell will be the URI of the session profile that will start the session, for example, **project://my_project/session_profiles/telnet_DUT6.ffsp**

  For instructions on configuring this value to be determined at runtime, see "Determining the device or session profile (dynamically) at runtime" on page 242.

- **iTest default session type**: This option starts a session without referencing a particular session profile. If you specify **iTest default session type**, then you must also (typically) specify additional property settings for the **open** step (for example, the IP address of the device). In this SNMP example, we hold the cursor over the error marker for the step

to learn that we need to specify the **IP address** for the device in the **SNMP Session Properties > SNMP MIB Browsers** properties group.



Select Session for Replay dialog box

iTest attempts to execute the captured items that you selected in the appropriate captured session. When iTest determines that a captured item can be executed in more than one session, it displays the **Select Session For Replay** dialog box to enable you to specify the session in which the items should be executed.

Select the session and then click **OK**.

Switch to Editor dialog box

Use this dialog box to specify the editor to activate (select), or to close, or to save the current editor contents.

Select Clean Editors	Add to the list any editors that are open, but have no content.
Invert Selection	Select editors that are currently unselected and unselect editors that are currently selected.
Select All	Select all editors in the list.
Activate Selected Editor	Applies only if a single editor is selected. Cause the editor that is selected in the list to become the current editor.
Close Selected Editors	Close all editors that are selected in the list. Save the current content of the editor that is selected in the list as an appropriate document.

Save Selected Editor	Applies only if a single editor is selected.
Show editors from all windows	Applies only if more than one instance of iTest is open. Display all editors

Workspace Launcher

Your workspace holds the projects in which you'll store all files that you generate using iTest. In some cases, you might want to access more than one workspace (for example, to access a coworker's test cases or to separate your local test environment from your workgroup's regression environment).

Note Any preferences that you set (**Window > Preferences**) apply to the current workspace only.

Default workspace

- ### Microsoft Windows systems

  The default workspace is **C:\Documents and Settings\userName\My Documents\iTest**. You'll create your working folders in this folder. So, when you create a working folder called **load_tests**, for example, the path to the new folder is

  ```
  C:\Documents and Settings\userName\My Documents\iTest\load_tests
  ```

- ### Linux and Unix systems

  The default workspace is **/home/username/iTest**. You'll create your working directories in this directory. So, when you create a working directory called **load_tests**, for example, the path to the new directory is

  ```
  /home/username/iTest/load_tests
  ```

Creating an alternative workspace

While a workspace is a directory hierarchy on a file system, it requires certain files to be in place to work properly. For this reason, it is easiest to copy an existing workspace to create an alternative to the default workspace.

1. Close iTest.
2. Copy the default workspace directory to the new location and rename it as needed.

   When you restart iTest, switch to the new workspace, as described here.

Switching to another workspace

1. Click **File > Switch Workspace**.
2. If you have already switched workspaces previously, then the previous workspaces appear for selection in the Switch Workspace menu. Either select a workspace from the list or select **Other**.
   - If you select a workspace from the list, iTest exits and then restarts using the selected workspace.

- If you select **Other**, then the **Workspace Launcher** dialog box opens. Browse to the workspace and then click **OK**.

3 **Copy Settings**: When you switch workspaces, you can transfer settings to the new workspace.

- **Workbench Layout**: Opened views, their size, and selected perspectives.
- **Working Sets**: The user defined working sets.

4 Click **OK**. iTest restarts and the iTest Explorer now displays the contents of the current workspace.

iTest Export wizard

You can use the Export wizard for a variety of tasks:

- Export your iTest preference settings so that you can share or use the settings in another iTest installation. See “Exporting and importing iTest preference settings” on page 861
- Export a iTest Capture report or test report in HTML format or text format so that you can share the information with someone that does not use iTest. See “Uses for test reports” on page 426.
- Export a iTest test report for use by another iTest user. See “Sharing test reports as files” on page 449.
- Export chart data into a standard format file so you or others can view the data using other applications (for example, Microsoft Excel or another data visualization application). See “Exporting chart data” on page 839.
- “Exporting test cases and other iTest documents” on page 809
- Exporting test reports for use by other iTest users. See “Exporting a test report as a compressed file (fftz format)” on page 451
- Advanced users: Export an Eclipse project set. See the Eclipse help.

Searching for files based on contents (File Search)

Specify search criteria to find files that contain specified text.

Filters dialog box

To specify which types of problem to display in the Problems view, click **Filter** in the toolbar.

Tips and Tricks

- The Spirent Community offers valuable information for new users and experts alike. Discussion forums, video tutorials, and example projects.
- For richer documentation on the progress of execution, check the box for 'Generate an execution message for each comment step that is executed' (on the 'General' page of the Test Case editor)
- If you have developed the perfect layout for your perspective, make it your personal custom perspective by clicking 'Window > Save Perspective As'

- You can drop single (or multiple) steps from the Capture view into your test case
- Every view and editor has 'Minimize' and 'Maximize' buttons in the upper right. When minimized, sizing tools appear in the upper left.
- After you create a rule using the analysis rule wizard, you can edit it as needed — right-click the rule and select 'Edit Analysis Rule'
- Use Ctrl+3 to open any view or perspective by typing its name
- Indent steps under Comment steps to organize your test case into functional modules — the comment text is the title of the module. You can click the 'Collapse All' button to work on one module at a time.
- Use $value in field replacements in analysis rule messages to display the value being validated
- If you can’t find a particular view, try clicking the 'Reset Layout' button
- Use the Capture Comments view to document your work as you capture (anything you type gets added as a Comment step in the test case).
- Comment text can include field replacements. This is an easy way to send data to the Execution view.
- Double-click a tab to maximize a view or editor. Double-click it again to minimize it.
- To improve performance, iTest does not map all items in very long responses. If you notice that the “blue boxes” do not appear in the later text of a response, you can increase the setting so that iTest evaluates more queries. Click 'Window > Preferences'. In the 'iTest' group, go to 'Response Mapping' and increase the 'Maximum number of queries to evaluate' setting.
- For a single item in the Capture view, Shift-drop the item into an active session window of the appropriate session type. iTest pastes the command into the session window, but does not execute it. This option enables you to edit the command before submitting it.
- Place a QuickCall library in the same folder as the session profile that it is associated with and with the same filename. For example, router3A.ffsp and router3A.fftc
- While working on QuickCall definitions on the Test Case editor 'Steps' page, click the Collapse All button to view only the QuickCall names and not the individual steps. You can then work on a single QuickCall definition without the clutter.
- Ctrl+Shift+Q is the same as clicking the QuickCall button, and the 'Execute a QuickCall' wizard is optimized for keystroke-only use so that manual testers need never use the mouse while performing a QuickCall.
- By default, arguments support runtime substitution of field replacement text. To disable substitution for an argument value, wrap the value inside { and } brackets. As a result, the argument text will be passed exactly as it appears and no substitution will occur.
- When iTest executes a QuickCall, the Query view lists the QuickCall library where the QuickCall is defined.
- Here is an easy way to add a 'run' step: While editing a test case, right-click the child test case in the Favorites view and select 'Insert step to run this test case'. The run step is added after the selected step.

- Add a 'comment' step and indent any number of related steps under the comment. You can then skip all of the steps by skipping only the comment step. In addition, you can collapse (fold) the comment step to temporarily hide the related steps to reduce clutter while editing.
- To view the description of any action, add a step and begin to type the action into the 'Action' cell. As you type, iTest displays a list of actions. When you select an action, iTest displays a brief description with a link to more complete help.
- You should know about a powerful alternative to procedures: The iTest QuickCall feature makes it easy to add custom actions to the built-in iTest actions (e.g., 'getTable' in SNMP sessions). You can use QuickCalls both during interactive (manual) testing and in test case steps.
- Comment steps are a great way to outline or pseudo-code a test case before adding actual steps. To add comments quickly, select a Comment step and then press Ctrl-Enter. Once your outline is complete, you can go back and insert commands.
- Use a Mail session and a 'response' field replacement to write the response to the 'summarize' action into the body of an email message
- You can use 'readFile' action to obtain text data and then use analysis rules for the step to save appropriate data items into variables for use later in the test case (using get, ${varName}, gget, or ${/data/varName}, as appropriate). In the text of the file, you might use delimiter characters (for example, commas or colons) to delimit data values to make extraction easier in the analysis rule.
- You can type escape sequences for non-printing characters directly into any command or property text.
- In any field that supports field replacements (including test case commands), the fastest way to insert a 'param' command is to right-click where the command should appear and then select 'Insert Parameter'.
- In any field that supports field replacements, the fastest way to insert a query on a stored response is to right-click where the query should go and select 'Insert > Query on Stored Response'.
- You can associate a response map with a procedure so that the response returned by the procedure will automatically have “blue boxes” — structured queries — available on the corresponding call step.
- On the Test Case editor 'Steps' page, click the 'Collapse All' button to remove clutter by displaying only the procedure names and not the individual steps.
- A 'write' step adds text into the response of a 'call' step. In a called procedure, you can use 'write' steps to include response text from multiple steps in the called procedure (as a multi-line string). The text that appears in the 'Description' cell of the 'write' step is appended to the response.
- You can use a 'return' step in the main procedure to exit a test case. Because you do not typically want to return every time the test case runs, you'll probably include the return step within an 'if' construct.
- A simple and powerful way to manually execute a series of CLI commands is to save the command text in a text file (for example, a Notepad text file). Copy the commands and then paste them into an active session window. The commands execute immediately.
- You have the option to save only selected sections of any report when you export it. See the help topic on “Customizing the content of a test report”.

- In iTest editors, Undo/redo (Ctrl-Z, Ctrl-Y) applies to any change.
- Use the iTestRT reporting plug-ins to publish reports in various ways.
- You can store a value from the response to a step (say, step 12). In a later step (say, step 19), you can add an analysis rule about a token in step 19 and compare its value to the value of the token extracted in step 12. So, for step 19, you can create an assertion like: $value = $tokenStep12 * 2
- To avoid unnecessary interruption while testing, after you have added all prompt definitions to a session profile or device definition, you can uncheck 'Learn prompts' in the 'Capture' properties group.
- The response map chaining feature enables you to specify that, during the mapping process, any response that does not find an applicable map in the specified response map library (or all applicable maps fail) should also check for applicable maps in one or more other libraries.
- You can use the last page of the 'New Response Map' wizard to have iTest take a first try at automatically generating a map for a response. (Click the 'Create a New Response Map' button in the Response view.)
- If you expect very large responses, allocate more memory to iTest. See the help topic on “Specifying how much memory to allocate to iTest”.
- While execution is paused, you can perform interactive actions in the session to view the results. If a step is useful, you might later add it to the test case. While execution is paused, the current step is marked in the Test Case editor with a yellow arrow.
- The Test Reports view offers an option to display only the 'latest' or most recent report for every test case. You might use this option while iteratively developing and executing a test case. When you finish, pick the last execution of each new test case that you developed that day and report the results.
- When you close a session with an exit command, the 'close' step is captured and (if so configured) the session window remains open. (To configure the session window to remain open after you disconnect a session, set the following two 'Execution' preferences appropriately: 'When execution finishes, close all Sessions windows' and, 'Before executing, close all active and inactive session windows'.)
- For an 'open' step in a test case, you have the option to override any of the property settings so that all steps in the session use the new property settings. Change any of the properties for the 'open' step that appears in the '<sessionType> Session Properties' section.
- Command Prompt : If an application does not run in a Command Prompt session, try running it on a Linux computer and using a Telnet session to test it.
- For Database sessions: If you are creating a step that you will eventually put into a loop, fetch records one-at-a-time. A record will be returned each time the loop encounters the step
- SNMP: To ensure good performance, add only the MIBs that you expect to use for testing.
- SNMP: On the 'Preferences' page, you can configure iTest to open the SNMP Traps view when a trap is received.
- SNMP: Clicking a node in the component tree is equivalent to populating the address bar (based on the tree node) and clicking 'Go'. The Application Under Test goes to the specified component.

- SNMP: Double-clicking a node in the component tree is equivalent to populating the address bar and then clicking 'Describe'.
- Use the text box at the top of the Structure view to test XPath expressions.
- Web: You can capture a snapshot of a Web page and the XML representation of the HTML code of the page. This feature is useful both for documenting the test case and for identifying targets when creating form maps. See the help topic on the Snapshot button.
- To locate a file or other resource quickly, press CTRL+SHIFT+R and begin typing the name

How to Contact Us

To obtain technical support for any Spirent Communications product, please contact our Support Services department using any of the following methods:

Spirent products and services

Information about Spirent Communications and its products and services appears on the company website at **https://support.spirent.com/SpirentCSC/**

Obtaining technical support

To obtain technical support for any Spirent Communications product, please contact our Support Services department using any of the following methods:

Americas

E-mail: **support@spirent.com**
Web: **https://support.spirent.com/SpirentCSC/**
Toll Free: +1 800-SPIRENT (+1 800-774-7368) (North America)
Phone: +1 818-676-2616
Hours: Monday through Friday, 05:30 to 18:00, Pacific Time

Europe, Middle East, Africa

E-mail: **support@spirent.com**
Web: **https://support.spirent.com/SpirentCSC/**
Phone: +33 (1) 6137 2270 (France)
Phone: +44 1803 546333 (UK)
Hours: Monday through Thursday, 09:00 to 18:00, Friday, 09:00 to 17:00, Paris Time

Asia Pacific

E-mail: **support@spirent.com**
Web: **https://support.spirent.com/SpirentCSC/**
Phone: +86 (800) 810-9529 (toll-free mainland China only)
Phone: +86 (10) 8233 0033 (China)
Hours: Monday through Friday, 09:00 to 18:00, Beijing Time

The Spirent Customer Service Center (https://support.spirent.com/SpirentCSC/) includes useful tools such as a powerful Knowledge Base with many Spirent iTest articles to serve your technical information needs. The Knowledge Base offers an easy-to-use browse mode along with an intelligent search that offers quick answers to your network analysis and measurement questions. New content is added daily by Spirent's communications and networking experts. Log in with your user ID and password to gain access to additional content that is available only to customers – user manuals, Help files, release notes, Tech Bulletins, and more. When you log in, you also gain access to download software and firmware, and to manage your SRs.

Company Address

Spirent Communications, Inc.
27349 Agoura Road
Calabasas, CA 91301 USA

Index

Symbols

A

B

C

D

E

F

G

H

I

J

Q

R

S

T

X

Z