



Inspired Innovation

Overview Manual

Spirent TestCenter™ Automation

January 2008

Spirent Communications, Inc.

26750 Agoura Road
Calabasas, CA
91302 USA

Copyright

© 2008 Spirent Communications, Inc. All Rights Reserved.

All of the company names and/or brand names and/or product names referred to in this document, in particular, the name "Spirent" and its logo device, are either registered trademarks or trademarks of Spirent plc and its subsidiaries, pending registration in accordance with relevant national laws. All other registered trademarks or trademarks are the property of their respective owners. The information contained in this document is subject to change without notice and does not represent a commitment on the part of Spirent Communications. The information in this document is believed to be accurate and reliable, however, Spirent Communications assumes no responsibility or liability for any errors or inaccuracies that may appear in the document.

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

About this Guide	5
Introduction	6
Related Documentation	6
How to Contact Us	8
Chapter 1: Introduction	9
Spirent TestCenter Automation	10
Spirent TestCenter Conformance	11
Key Benefits of Spirent TestCenter Automation	11
Automation in the Spirent TestCenter Environment	11
The Spirent TestCenter API	13
This Manual	14
Chapter 2: The Spirent TestCenter Data Model	15
The Network Test Environment	16
The Software Test Environment	17
Common Object Types	18
Creating a Spirent TestCenter Test	20
General Steps to Set Up and Run Tests	20
A Spirent TestCenter Test (Traffic Generation and Analysis)	21
Chapter 3: Using the Spirent TestCenter API	23
Test Elements and Test Execution	24
Test Configuration Objects	24
Relations	24
Commands and Command Objects	24
The API Functions	25
Spirent TestCenter API Syntax	26
Object Handles	28
Creating Object Handles	28
Using Object Handles	28
Direct Descendant Notation	29
Object Attributes	29
Setting Attributes	29
Retrieving Attribute Values	30
Descendant-Attribute Notation	30
Automatic Object Creation	30
Using Relations	31
Test Output	32

Spirent TestCenter Test Results	32
Data Capture	34
Spirent TestCenter Automation Run-Time Environment	35
Creating Test Configurations	35
Using the Tcl Interface (An Example).	36
Initialization	36
Chassis Connection and Port Reservation	37
Project and Port Objects	37
Traffic Configuration	38
Result Subscription	39
Test Execution	41
Cleanup	42
Test Completion	43
Appendix A: Spirent TestCenter Licenses	45
Index	47

About this Guide

In About this Guide...

- [Introduction 6](#)
- [Related Documentation 6](#)
- [How to Contact Us 8](#)

Introduction

This manual provides an overview of Spirent TestCenter Automation and how to use the Spirent TestCenter API to create a test configuration, run the test, and retrieve test results. The manual includes a description of the Spirent TestCenter software packages, the object model, and the Tcl functions in the API.



Note: This manual is not intended as a reference guide for any of the Spirent TestCenter Automation packages. For detailed information on the features, functions, operations, attributes, and methodology of the software for your particular Spirent TestCenter package, refer to the document set on the CD with your Spirent TestCenter Automation software.

This manual is intended for users who wish to use the Spirent TestCenter Automation API to perform tests using the features of one of the Spirent TestCenter Automation packages (such as the Multicast base package or the RFC 2544 test package). It is assumed that users of this manual have the following knowledge and experience:

- Familiarity with the operating system environment on your PC or workstation (Microsoft® Windows® or Linux®/Unix®).
- Moderate familiarity with Spirent TestCenter equipment
- Working knowledge of data communications theory and practice
- Ability to program with the Tcl scripting language.

Related Documentation

Additional documentation items that are related to this manual are listed below.

- *Spirent TestCenter Automation Object Reference*. Contains reference information about the objects in the *Spirent TestCenter* data model.
- *Spirent TestCenter Automation Programmer's Guide*. Describes the Spirent TestCenter Automation functions and contains information about using the API to write Tcl scripts for specific protocols and particular elements of network performance testing.
- *Spirent TestCenter Conformance Automation Object Reference*. Contains reference information about the objects in the *Spirent TestCenter Conformance* data model.
- *Spirent TestCenter Conformance Automation Programmer's Guide*. Describes how to use Spirent TestCenter Conformance Automation to write Tcl scripts for conformance and interoperability tests on Spirent TestCenter.
- *External Time Reference (GPS/CDMA) User Guide*. Describes the GPS/CDMA kits available from Spirent Communication, explains GPS/CDMA theory of operation, and describes how to set up GPS/CDMA for use with Spirent TestCenter and SmartBits equipment.

- *Getting Started with Spirent TestCenter* provides hardware set up, software installation, and licensing instructions for Spirent customers who are receiving and installing new Spirent TestCenter chassis.
- *Spirent TestCenter System Reference Manual*. Describes the Spirent TestCenter chassis, modules, accessories, and software applications. General information is also provided on system administration functions, testing procedures, and diagnostics.

In addition to these manuals, all Spirent TestCenter software applications include detailed, context-sensitive online Help systems.

A glossary of Spirent TestCenter terminology is available in each Spirent TestCenter online Help file.

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:

Americas

E-mail: support@spirent.com

Web: <http://support.spirentcom.com>

Toll Free: +1 800-SPIRENT (+1 800-774-7368) (US and Canada)

Phone: +1 818-676-2616

Fax: +1 818-880-9154

Hours: Monday through Friday, 05:30 to 16:30, Pacific Time

Europe, Africa, Middle East

E-mail: support@spirent.com

Web: <http://support.spirentcom.com>

Phone: +33 (0) 1 61 37 22 70

Fax: +33 (0) 1 61 37 22 51

Hours: Monday through Thursday, 09:00 to 18:00, Friday, 09:00 to 17:00, Paris Time

Asia Pacific

E-mail: supportchina@spirent.com

Web: <http://support.spirentcom.com.cn>

Phone: 400 810 9529 (mainland China)

Phone: +86 400 810 9529 (outside China)

Fax: +86 10 8233 0022

Hours: Monday through Friday, 09:00 to 18:00, Beijing Time

The latest versions of user manuals, application notes, and software and firmware updates are available on the Spirent Communications Customer Service Center websites at <http://support.spirentcom.com> and <http://support.spirentcom.com.cn> (China).

Information about Spirent Communications and its products and services can be found on the main company websites at <http://www.spirentcom.com> and <http://www.spirentcom.com.cn> (China).

Company Address

Spirent Communications, Inc.
26750 Agoura Road
Calabasas, CA 91302
USA

Chapter 1

Introduction

In this chapter...

- [Spirent TestCenter Automation 10](#)
- [Automation in the Spirent TestCenter Environment 11](#)
- [The Spirent TestCenter API 13](#)
- [This Manual 14](#)

Spirent TestCenter Automation

Spirent TestCenter Automation is an automated software system for network performance analysis. Spirent TestCenter Automation provides a Tcl-based Application Programming Interface (API) that you use to create and run tests. You use Spirent TestCenter software together with a network hardware configuration that includes Spirent TestCenter hardware and your network device(s). The Spirent TestCenter software/hardware combination generates test traffic to measure the performance of your network device.

Spirent Communications distributes Spirent TestCenter Automation software as part of the Spirent TestCenter software product. The Spirent TestCenter Automation capabilities are available in the set of Spirent TestCenter software *packages* that support network protocols and RFC test methodologies. There are two classes of Spirent TestCenter packages:

- The *Base* packages provide software for testing network protocols.
- The *Test* packages provide software for testing based on well-defined test methodologies that are either RFC-based standards or developed by Spirent in working with its customers.

To locate product datasheets that describe these packages, use the **Spirent Document Finder**. The Document Finder is located here:

<http://www.spirentcom.com/about/index.cfm?media=7&ws=327>

To use the Spirent TestCenter packages, you must obtain the appropriate license(s). (For information about licenses, see *Appendix A, “Spirent TestCenter Licenses.”*) For information about installing Spirent TestCenter software, refer to the *Getting Started with Spirent TestCenter* document.

The Spirent TestCenter software includes a graphical user interface (GUI) for the test system. The Spirent TestCenter GUI is separate from the Spirent TestCenter Automation software. The GUI allows you to generate tests, run the tests, and review the results without writing any scripts or programs. These GUI interfaces are both easy to use and powerful, but many users require the ability to develop unique tests or tests that will run unattended for hours or days. Spirent TestCenter Automation also supports easy customization of tests. In general, the automation interface and the GUI provide the same capabilities.

Spirent TestCenter Conformance

Spirent Communications provides automation software for Spirent TestCenter Conformance Application. Spirent TestCenter Conformance Application executes Spirent Communications conformance and interoperability test suites on Spirent TestCenter.

- A *conformance test suite* verifies the compliance of an implementation under test (IUT) to a standard or specification.
- An *interoperability test suite* verifies how two systems work together in accordance with a standard or specification.

Spirent TestCenter Conformance Automation defines a set of commands that you use to manipulate the components of a test suite configuration. You use Spirent TestCenter Conformance Automation in the context of the Spirent TestCenter Automation Application Programming Interface (API) to create and then run the test suite.

For information about using Spirent TestCenter Conformance Automation, see the *Spirent TestCenter Conformance Automation Programmer's Guide*.

Key Benefits of Spirent TestCenter Automation

<i>Test automation</i>	You can create complex tests with numerous repetitions and virtually unlimited test durations. You can automate your results verification, because Spirent TestCenter Automation allows access to both the results and the test configurations while the program is running.
<i>Single interface</i>	Spirent TestCenter Automation packages share a common architecture, syntax, and set of functions. The Spirent TestCenter API provides access to the complete set of test capabilities.
<i>Default values</i>	Spirent TestCenter Automation provides an extensive set of default values for test attributes.
<i>Tcl support</i>	Spirent TestCenter Automation provides a Tcl interface to the Spirent TestCenter Automation tools.
<i>Interactive Testing</i>	Spirent TestCenter Automation supports interactive access to the test environment. You can examine test results and modify the test configuration for a running test.

Automation in the Spirent TestCenter Environment

When you use Spirent TestCenter Automation, you provide test information by using a set of Tcl functions to create a representation of a test configuration.

Figure 1-1 on page 12 shows the relationships between the test information that you provide, Spirent TestCenter Automation, and the Spirent TestCenter core system (the Spirent TestCenter software, firmware and system hardware).

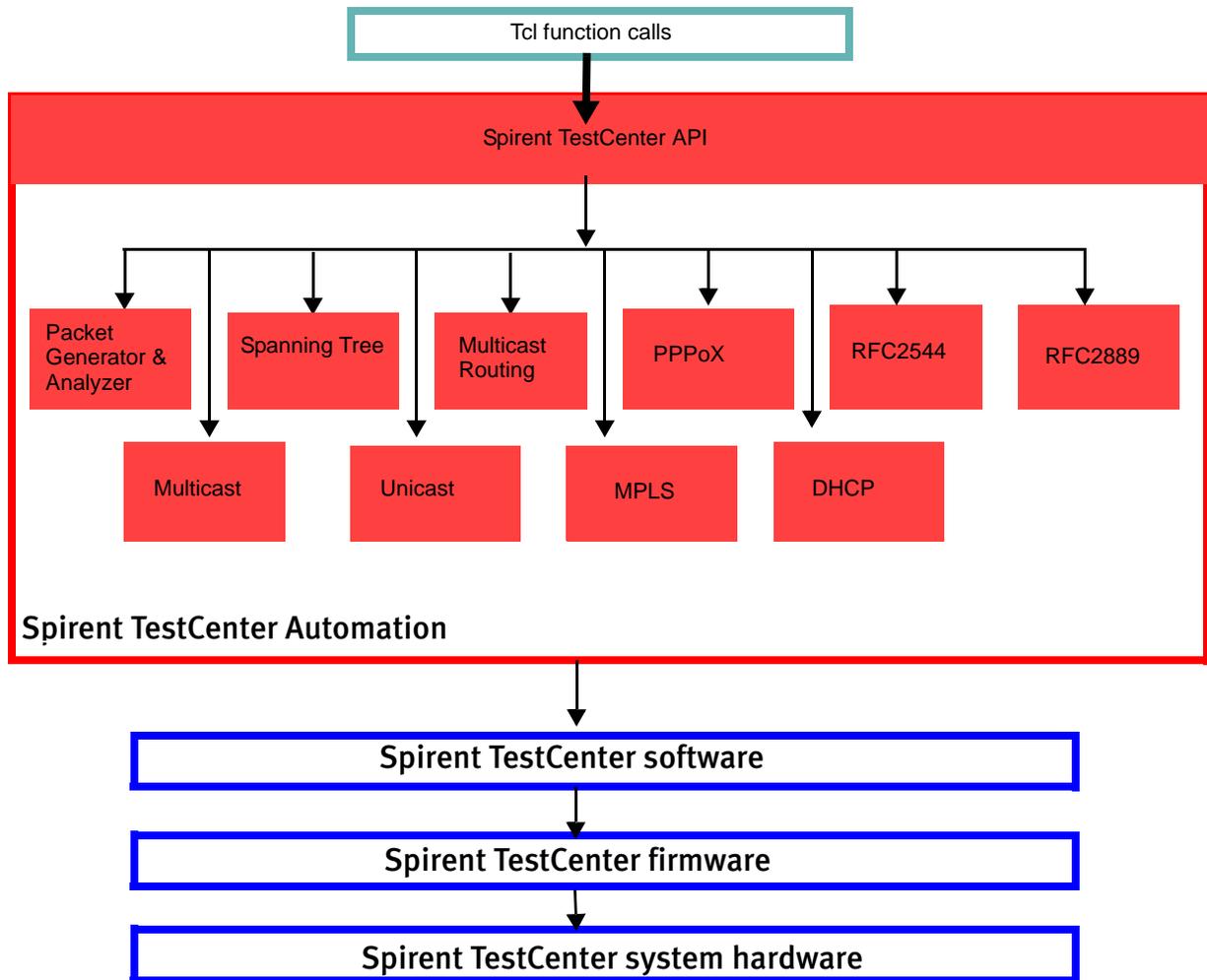


Figure 1-1. Spirent TestCenter Automation and the Spirent TestCenter Environment

You use the Spirent TestCenter API to create and run the test, and to retrieve the test results. Spirent TestCenter Automation handles the communication with the firmware on the Spirent TestCenter hardware (chassis and cards).

The Spirent TestCenter API

Spirent TestCenter Automation provides an API that you use to create and run tests, and to retrieve the test results. The API defines a set of functions that you can call from a Tcl script or from a Tcl shell window.

The API consists of a common set of functions that are used for all of the Spirent TestCenter Automation packages. This common set of functions reflects the architecture of the Spirent TestCenter packages. Because the packages share a common architecture and interface, once you have learned how to write a test program for one package, it is easy to write tests for the other packages.

The API supports all of the features provided by Spirent TestCenter. It provides the capability to run tests for virtually unlimited test durations, and it gives you access to the test configuration and test results during execution.

Figure 1-2 shows what your application provides to Spirent TestCenter Automation (a test description) and what Spirent TestCenter Automation produces (test results). (Descriptions of items 1 - 4 in the figure begin on [page 14](#).)

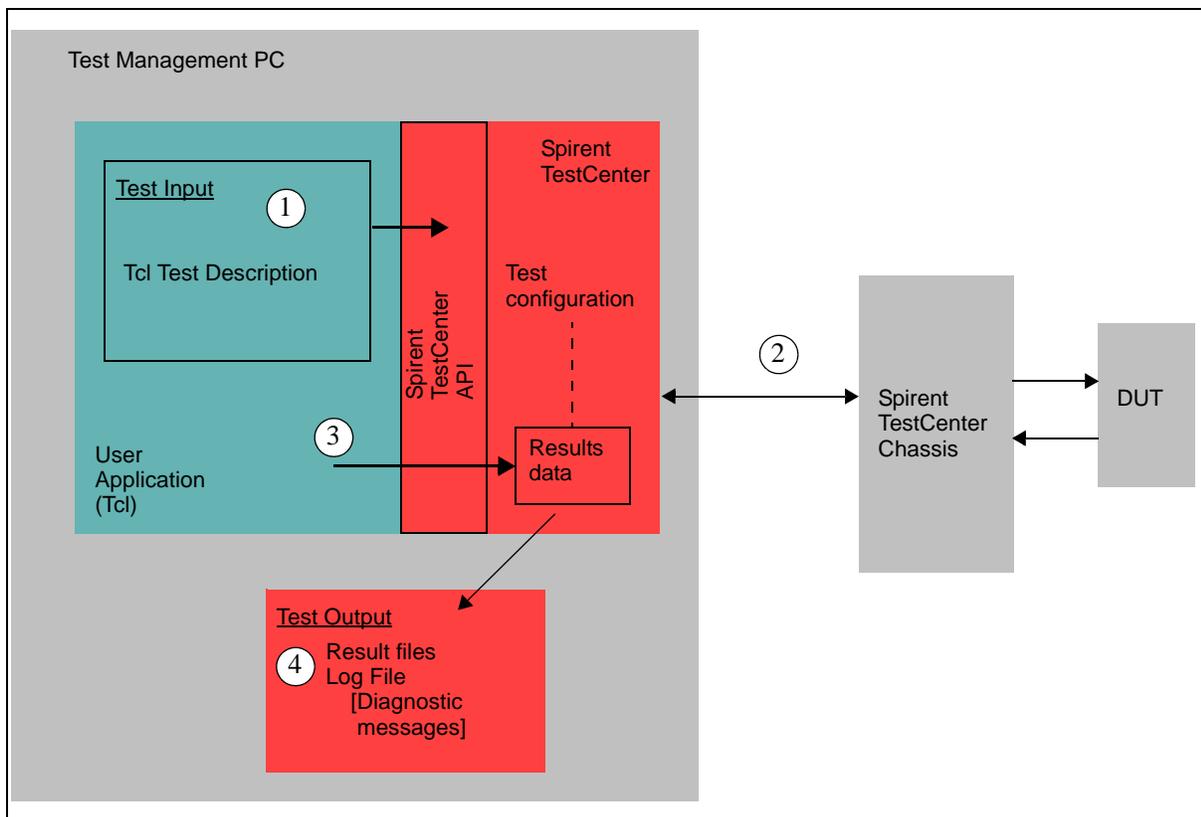


Figure 1-2. Using Spirent TestCenter Automation

- 1** Test input — Use the Spirent TestCenter API to define your test configuration. To create a test description, you can use either a Tcl script or you use a Tcl shell window. If you use a script, you place Tcl function calls to the API in your script, and then run the script in a Tcl shell. Or, you can enter the function calls directly in the Tcl shell window.
- 2** Test execution — Based on your test description, Spirent TestCenter Automation creates an internal representation of the test configuration. Spirent TestCenter Automation runs the test, handling the communication with the Spirent TestCenter Chassis and directing the test execution.
- 3** Results data — Depending on context, results data are available to your application through the Spirent TestCenter API or in output files. Real-time results are available during the test, and Spirent TestCenter compiles results at the end of the test.
- 4** Test output — Spirent TestCenter Automation produces several kinds of output, including results files and a log file that contains diagnostic messages.

This Manual

The remaining chapters in this manual provide the following information:

- *Chapter 2, “The Spirent TestCenter Data Model,”* describes the Spirent TestCenter object model and how your tests use it in the Spirent TestCenter network performance analysis system.
- *Chapter 3, “Using the Spirent TestCenter API,”* provides information about how to use the API functions to create and use Spirent TestCenter Automation objects to run your tests. This chapter also provides information about the different kinds of test output, and how to use the Spirent TestCenter Automation API to access test results.

Chapter 2

The Spirent TestCenter Data Model

Spirent TestCenter Automation defines a data model consisting of objects that represent the different components of a test configuration. When you define a test, you use a subset of the objects to create an object hierarchy that represents your particular test configuration.

Spirent TestCenter Automation uses the object hierarchy to run your test; the attributes for the objects provide the information that is required. For example, if you are testing a BGP configuration using IPv4, you provide required data for BGP update messages by setting the **Origin** and **NextHop** attributes for a **BgpIpv4RouteConfig** object.

In addition to the objects that describe a test configuration, the data model also includes objects that define result data. When Spirent TestCenter collects test results, it sets the value of the corresponding result attributes. (In addition to providing results through the API, Spirent TestCenter provides additional results in output files.)

In this chapter...

- [The Network Test Environment 16](#)
- [The Software Test Environment 17](#)
- [Common Object Types 18](#)
- [Creating a Spirent TestCenter Test 20](#)
- [A Spirent TestCenter Test \(Traffic Generation and Analysis\) 21](#)

Chapter 3, “Using the Spirent TestCenter API” contains an example Tcl script. This example is intended to demonstrate the use of the Spirent TestCenter data model.

The Network Test Environment

Spirent TestCenter objects are software definitions that represent the elements of an emulated network test configuration. The Spirent TestCenter software runs in the Spirent TestCenter network test environment, which includes:

- one or more Spirent TestCenter modules
- one or more Devices Under Test (DUTs).

This manual uses an example of a simple network configuration to show how the object model is applied to the Spirent TestCenter test environment. *Figure 2-1* shows a test environment that contains a test management PC running Spirent TestCenter, a Spirent TestCenter chassis containing a single Spirent TestCenter module with two ports. The example test is configured to run a back-to-back test, in which one port on the module transmits to the other (receiving) port.

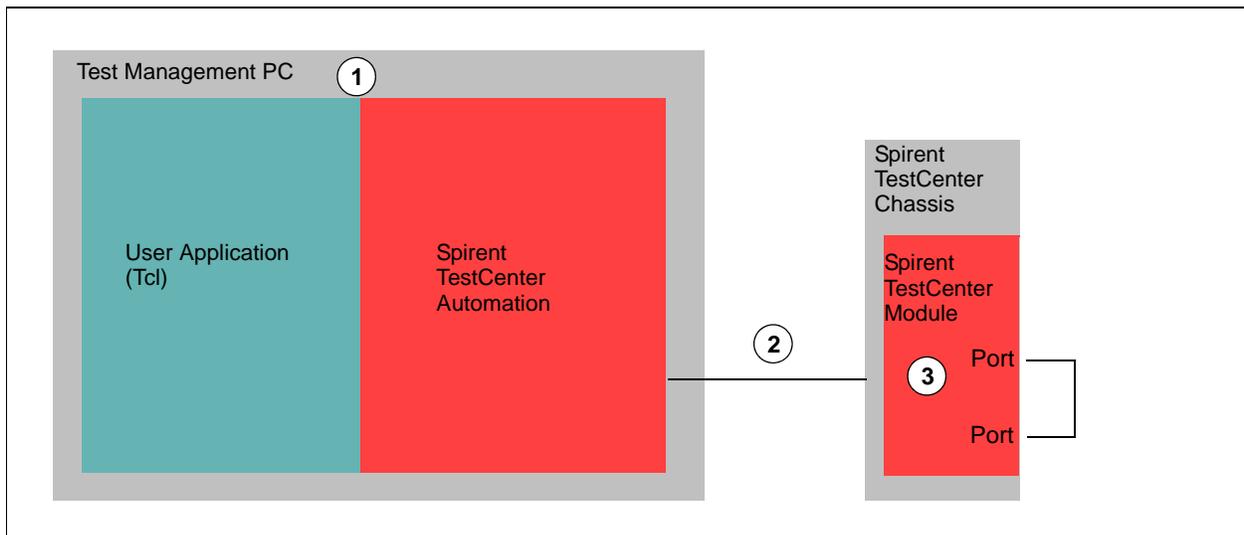


Figure 2-1. A Spirent TestCenter Test Environment

As shown in *Figure 2-1*, Spirent TestCenter Automation (along with your application) runs on a PC (1). The connection between the PC and the Spirent TestCenter chassis (2) provides Spirent TestCenter Automation access to the chassis (and thus the capability of running the test), but it is not part of the test configuration.

- The *hardware* test configuration is defined by (3) a Spirent TestCenter module and its ports, along with any DUTs connected to the ports. The example used in this manual is a simple back-to-back configuration that is intended to provide an overview of using Spirent TestCenter Automation. In other configurations, the ports would be connected to one or more DUTs.
- The *software* test configuration is defined by your application. The Spirent TestCenter Automation software provides you with the means of superimposing a software model (the object hierarchy) on top of the hardware test configuration.

Spirent TestCenter uses this software model to emulate a network environment; the emulated network can contain hundreds or thousands of network hosts, producing a simulation of real network traffic.

The Software Test Environment

In the Spirent TestCenter performance analysis system, your application uses Spirent TestCenter Automation software to create a test, run the test, and retrieve the test results.

[Figure 2-2](#) shows a general representation of the process of creating a test.

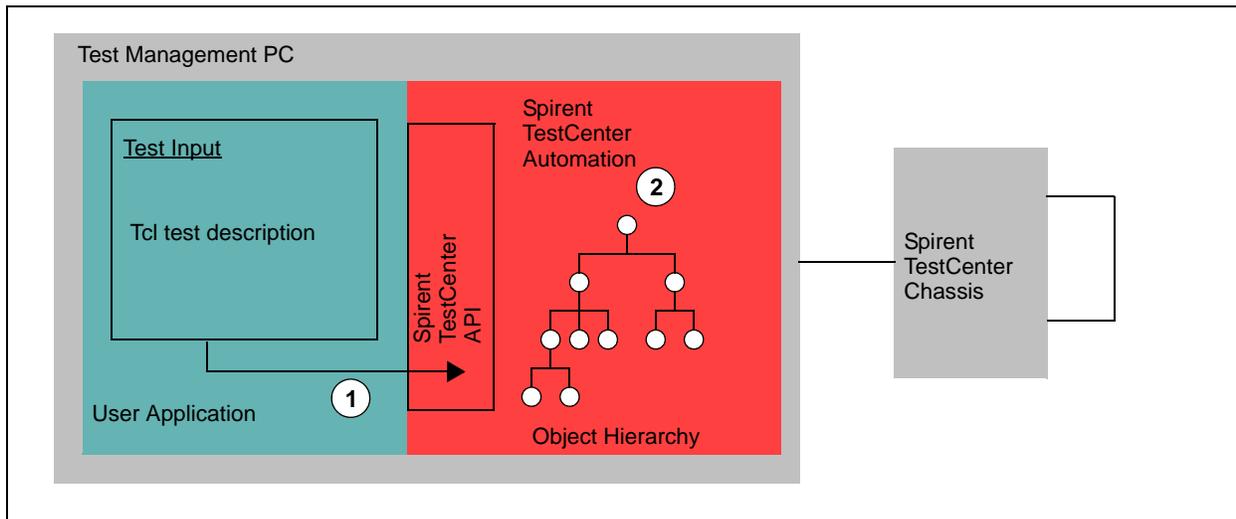


Figure 2-2. Creating a Spirent TestCenter Automation Test

When you create a test, your Tcl application uses the API to provide test input to Spirent TestCenter Automation (1). Spirent TestCenter Automation creates the object hierarchy (2), which it will use to run the test. Before running the test, Spirent TestCenter Automation will validate the configuration described in the object hierarchy. If you have created a valid configuration, Spirent TestCenter Automation will run the test.

[Figure 2-3 on page 18](#) shows the effect of running the test.

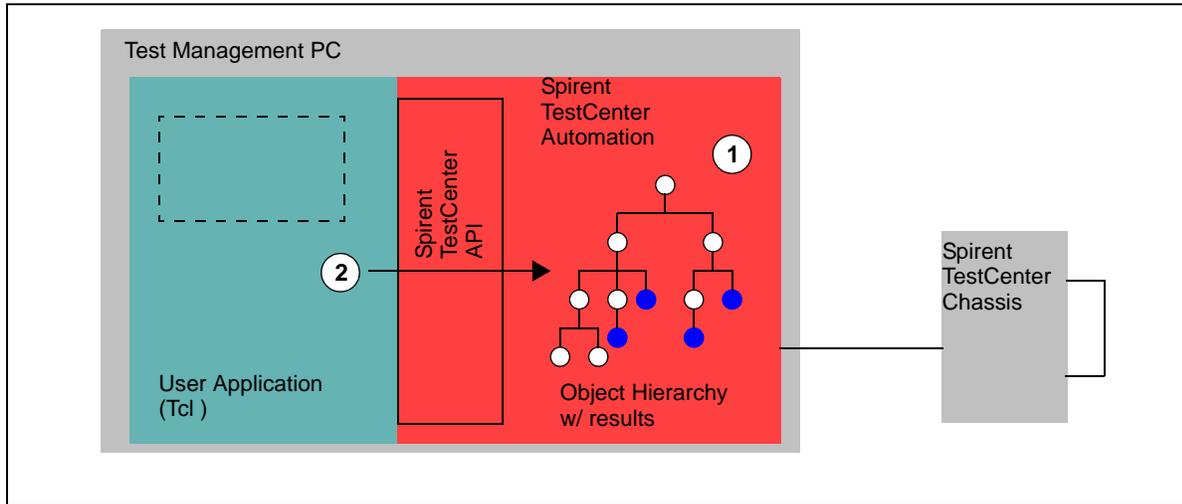


Figure 2-3. Running a Spirent TestCenter Automation Test

Spirent TestCenter maintains the object hierarchy for the duration of the test session. During the test, Spirent TestCenter collects test results and stores the data in the hierarchy (1 – in the figure, objects are rendered in blue to indicate result objects that have been created, and result attributes that have been updated). Your application can retrieve the result attribute values and manipulate the objects in the hierarchy during test execution (2). This gives you considerable control over the test environment.

Common Object Types

The network connections and traffic in your test environment are represented by two basic components in Spirent TestCenter testing: ports and traffic. Spirent TestCenter defines objects to represent these components and other components of a test configuration. The particular set of object types and the number of objects that you use for your test is determined by the Spirent TestCenter test that you are running and the way you structure your test – how many ports you use and how you define the traffic in your test.

When you create a test configuration, you use Spirent TestCenter objects to create an object hierarchy. [Table 2-1 on page 19](#) shows some of the object types that are common to many test configurations. In the table, the type names indicate specific object types; you use these names when you create objects.

The particular set of objects that you use for a test is determined by the test type and protocol. (For a diagram of the complete object hierarchy, and a complete description of object types and attributes, see the *Spirent TestCenter Automation Object Reference Manual*.)

Table 2-1. Common Spirent TestCenter Object Types

Object Type (Examples)	Purpose	Attributes (Examples)
Project	A Project object is the root of the object hierarchy. Every test configuration must have a project object.	Active Name
Port	A Port object is a child of the project object. You create a Port object to identify a port on a Spirent TestCenter module.	Active Location Name
Host	A Host object is a child of the Project object. You create a Host object to emulate a host system.	Active DeviceCount
StreamBlock	A StreamBlock object defines the characteristics for a stream of network traffic.	Active FixedFrameLength InterFrameGap
EthernetII	An EthernetII object is an example of a header object. An EthernetII object defines the data for an Ethernet frame in network traffic.	dstMac etherType preamble srcMac
Router	A Router object defines an emulated router.	Active DeviceCount RouterId
BgpRouterConfig	A BgpRouterConfig object is an example of a router configuration object. A BgpRouterConfig object defines the characteristics of a BGP router for the emulated network environment.	Active GracefulRestart HoldTimeInterval IpVersion KeepAliveInterval

Creating a Spirent TestCenter Test

When you use Spirent TestCenter Automation to create a test, you create an object hierarchy that will represent your test configuration. When you create Spirent TestCenter objects, you also set attribute values for the objects. The attributes provide information that determines the test characteristics. For example, **Port** objects have the attribute **Location**; for a **Port** object, you set the **Location** attribute to identify a Spirent TestCenter chassis, a slot containing a Spirent TestCenter module, and a port on the module.

When you create a Spirent TestCenter Automation test, you use the Spirent TestCenter API to define your test. The API provides functions that you use to create and manipulate the objects in the hierarchy. ([Chapter 3](#) describes how to use the API.)

General Steps to Set Up and Run Tests

[Table 2-2](#) lists the general steps required to set up and run a test.

Table 2-2. General Steps to Set up and Run Tests

Step
1 Set up a communication link between your PC and your Spirent TestCenter chassis.
2 Prepare the DUT/SUT.
3 Connect the Spirent TestCenter chassis to the DUT/SUT.
4 Initialize the Spirent TestCenter API to establish the object set context.
5 Create a project object and set the project attributes.
6 Create Port objects and set the port attributes.
7 Create StreamBlock objects (and, if necessary, header objects for the traffic), and set the appropriate attributes.
8 Set up the Spirent TestCenter generator and analyzer for traffic support.
9 Establish the software connection from your PC to the Spirent TestCenter chassis.
10 Start the test.
11 Review/Retrieve the test results.
12 Cleanup after the test has completed.

To review test results (step 11), you can display the contents of results files using a spreadsheet program, such as OpenOffice Calc, Gnumeric, or Microsoft[®] Excel[®]. To retrieve the test results and use the data in your program, use the API to access the result objects. (See [Chapter 3](#) for information about test output.)

A Spirent TestCenter Test (Traffic Generation and Analysis)

The example in the following chapter is based on the simple network configuration described in “*The Network Test Environment*” on page 16.

Figure 2-4 shows an object hierarchy for this test environment. The configuration for this test uses two ports; traffic with the appropriate headers will be generated from the transmitting port.

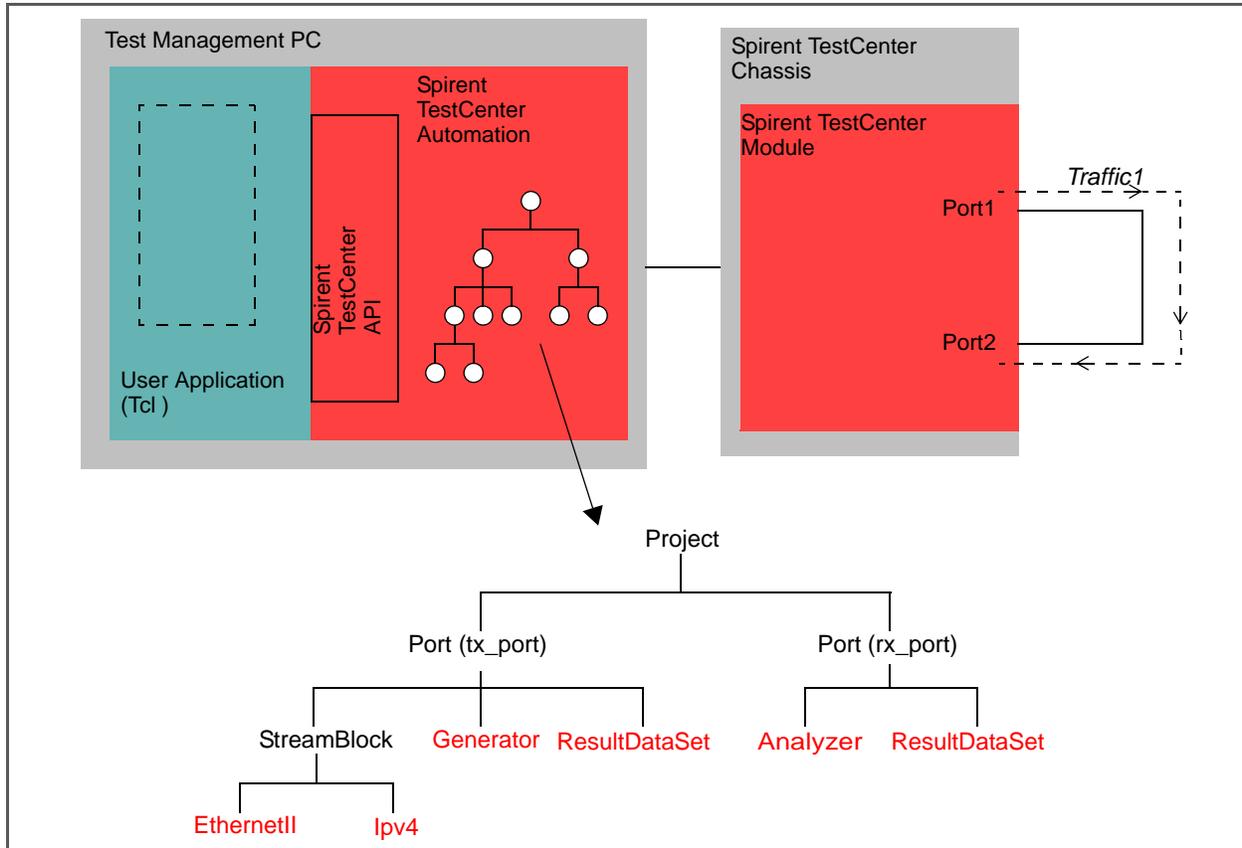


Figure 2-4. Spirent TestCenter Test Object Hierarchy

Figure 2-4 shows a representation of the object model that you create by calling functions in the Spirent TestCenter Automation API. The object names that are rendered in red indicate objects that Spirent TestCenter creates automatically. (Chapter 3 describes how you use automatically created objects.)

A Spirent TestCenter object hierarchy is organized by a specific set of rules:

- The root of the hierarchy is a *Project* object.
- When you create objects, you must create a parent object before you create any of its children. For a particular (parent) object type, the set of object types that are allowed as its children is limited. For information about what object types are allowed, see the *Spirent TestCenter Automation Object Reference Manual*.



Chapter 3

Using the Spirent TestCenter API

In this chapter...

- [Test Elements and Test Execution 24](#)
- [The API Functions 25](#)
- [Spirent TestCenter API Syntax 26](#)
- [Object Handles 28](#)
- [Object Attributes 29](#)
- [Automatic Object Creation 30](#)
- [Using Relations 31](#)
- [Test Output 32](#)
- [Spirent TestCenter Automation Run-Time Environment 35](#)
- [Creating Test Configurations 35](#)
- [Using the Tcl Interface \(An Example\) 36](#)
- [Test Completion 43](#)

Test Elements and Test Execution

When you use Spirent TestCenter Automation to run a test, you write a Tcl script that uses the following elements:

- *Test Configuration Objects*
- *Relations*
- *Commands and Command Objects.*

Test Configuration Objects

Test configuration objects describe the components of your test configuration. These objects provide the data that Spirent TestCenter needs to create and run a test. Examples of test configuration objects are:

Project	StreamBlock	BgpRouterConfig
Host	EthernetII	BgpIpv4RouteConfig
Port	Vlans	OspfV3RouterConfig
Router	Vlan	OspfV3RouterLSA

Relations

Relations define the connections between the objects in your test configuration. Every object is connected to at least one other object by a **ParentChild** relation. Spirent TestCenter creates **ParentChild** relations automatically when you create objects. In other cases, you must create relations to support specific test operations. For example, the **ExpectedRxPort** relation connects a **StreamBlock** object to a **Port** object, to identify the port that will receive traffic.

Commands and Command Objects

Spirent TestCenter Automation uses command objects to define test actions such as **StartProtocol**, **CaptureStart**, and **L2LearningStart**. The command parameters are defined as attributes for the associated command objects.



Note: Command names and command objects names are slightly different. Command object names have the suffix “Command” added to the command name.

There are two methods of invoking commands:

- 1 You can use the **perform** function to invoke a command. For example:

```
stc::perform generatorStart -generatorList $generator
```

When you use the **perform** function, you specify the command name, along with name-value pairs for attributes that are defined for the corresponding command object. When you use **perform**, you must call the function each time you want to execute an action.

- 2 You can create command objects and use the sequencer to execute a set of commands. When you use the sequencer, you add the set of command objects to the sequencer, and then call the **perform** function to execute the **SequencerStart** command. (For information about using the sequencer, see the *Spirent TestCenter Automation Programmer's Guide*.)

Test Execution (Traffic Generation and Analysis)

The basic operations of test execution are traffic generation and analysis. Spirent TestCenter Automation defines **Generator** and **Analyzer** objects to support these operations. (Spirent TestCenter creates these objects automatically.)

Once you have created the objects and relations for your test configuration, use the following commands to control generation and analysis:

- **AnalyzerStart**
- **AnalyzerStop**
- **GeneratorStart**
- **GeneratorStop**

See the Tcl script that is described beginning on [page 36](#) for an example of using the analyzer and generator components.

The API Functions

The Spirent TestCenter API provides a set of functions that you use to create and run tests. These functions allow you to do the following:

- Create objects to build the object hierarchy for your test configuration.
- Set the value of object attributes to define the characteristics of your test.
- Get the value of object attributes, including result attributes.
- Run the test after it has been created.
- Connect to your Spirent TestCenter chassis.

[Table 3-1 on page 26](#) shows the list of API functions. You use these functions to create and run tests with any Spirent TestCenter test configuration. (See the *Spirent TestCenter Automation Programmer's Guide* for a complete description of these functions.)

Table 3-1. API Functions

Function	Description
apply	Applies a test configuration to the Spirent TestCenter firmware.
config	Sets or modifies the value of an attribute.
connect	Establishes a connection with a Spirent TestCenter chassis.
create	Creates an object in a test hierarchy.
delete	Deletes an object in a test hierarchy.
disconnect	Removes a connection with a Spirent TestCenter chassis.
get	Retrieves the value of an attribute.
help	Displays help text in the Tcl window.
log	Writes a diagnostic message to the log file.
perform	Invokes an operation.
release	Releases a port group.
reserve	Reserves a port group.
sleep	Suspends application execution.
subscribe	Directs result output to a file or to standard output.
unsubscribe	Removes a subscription.
waitUntilComplete	Suspends your application until the sequence of commands has finished executing.

Spirent TestCenter API Syntax

This section provides an overview of the API syntax. For a more complete description of the Spirent TestCenter Automation syntax, see the *Spirent TestCenter Automation Programmer's Guide*.

The Spirent TestCenter Automation functions use a standard syntax that allows you to perform the operation of the function and, if appropriate, specify attribute settings at the same time. For example, you can create an object and set the attribute values for the object with a single call to the **create** function:

```
set hPort [stc::create Port -under $hProject \  
-location $chassis/$slot/$port]
```

This statement creates a Port object (as a child of a Project object), and specifies the location (chassis, slot, and port) in the same function call.

The syntax for a call to a Spirent TestCenter Automation function is as follows:

```
functionName [objectReference] [-attributeReference [...]]
```

To use a Spirent TestCenter Automation function, specify the name of the function, followed by any parameters that may be required by the particular function. The set of parameters may include an object reference and one or more attribute references.

- Object references are either an object type name, the handle of an existing object, or a path name that identifies a set of object types or an object. For example, when you call the **create** function to create an object, the object reference is the type of object to be created. When you call the **config** function to set attribute values, the object reference is the handle of the object you are modifying. (See “*Object Handles*” on page 28 for information about how to use object handles.) For a complete description of object references, see the syntax discussion in the *Spirent TestCenter Automation Programmer’s Guide*.
- An attribute reference can be an attribute name, an attribute name-value pair, a path name, or a relation reference. To specify the basic form of an attribute reference (an attribute name-value pair), use the following format:

```
-attributeName attributeValue
```

The attribute name must start with a dash character (-). The attribute value is separated from the name by a space.

Using the example given above:

```
create Port -under $hProject -location $chassis/$slot/$port
```

- The object reference `Port` specifies the type of object that you are creating.
- The attribute **-under** specifies the object handle (`hProject`) for the parent of the newly created object.
- The attribute **-location** specifies the chassis-slot-port combination (`$chassis/$slot/$port`).



Note: The Spirent TestCenter API is referenced through the namespace “stc.” It is recommended that you use the namespace syntax in your scripts to identify Spirent TestCenter Automation functions (for example, `stc::create`).

Object Handles

In many cases, when you call Spirent TestCenter Automation functions, you either provide or obtain an object *handle*. A handle is a value that identifies a Spirent TestCenter Automation object. Object handles are the links between your application and the object hierarchy maintained by Spirent TestCenter Automation. The following sections describe how to create and use object handles.



Note: In some circumstances, Spirent TestCenter will create objects on your behalf. For information about using these objects, see [“Automatic Object Creation” on page 30](#).

Creating Object Handles

Spirent TestCenter Automation creates handles when it creates the objects in your object hierarchy. When you call the **create** function, the object handle is the return value from the function:

```
set project [stc::create Project -testName "TrafficTest"]
```

When the **create** function returns, the variable *project* is set to the value of the object handle for the newly created **Project** object. Your application then uses the object handle in other calls to Spirent TestCenter Automation functions.

Using Object Handles

You use object handles in the following circumstances:

- *To identify the parent for a new object.* With the exception of the **Project** object, you must specify a parent when you create an object. For example:

```
set port_handle [stc::create Port -under $project_handle]
```

In this example, the **-under** attribute specifies the **Project** object handle as the parent for the new **Port** object. The **create** function returns the handle for the **Port** object; you use this object handle as the parent for Traffic objects or other objects that are children of an Ethernet port object.

- *To gain access to object attributes.* You provide a handle when you call **get** to retrieve an attribute value, or when you call **config** to modify an attribute value. You also use handles when you perform an operation (for example, when you call the **perform** function). See [“Object Attributes” on page 29](#) for examples of using the **get** and **config** functions.

Direct Descendant Notation

Spirent TestCenter Automation syntax supports a path notation called Direct Descendant Notation (DDN). An object reference that uses DDN begins with an object handle followed by an object path (object type names separated by periods). For example, the following call to the `config` function modifies the `-location` attribute for the first `Port` child of the `$project` object.

```
stc::config $project.Port -location "mychassis1/1/2"
```

DDN specification allows you to reference attributes of descendant objects without retrieving additional object handles. For a complete description of Direct Descendant Notation, see the *Spirent TestCenter Automation Programmer's Guide*.

Object Attributes

Objects represent the components of your test configuration. Object attributes specify the characteristics of a particular type of object. For example, the `StreamBlock` object attribute `fixedFrameLength` specifies the number of bytes in a frame for a stream of data to be transmitted over the test network. In the course of creating a test configuration and running a test, you will set and retrieve attribute values.

Setting Attributes

You set or modify object attributes under the following circumstances:

- When you create an object, its attributes have default values. You can override the default values by setting attribute values in the call to the `create` function. For example:

```
set tx_port [stc::create Port -under $project \  
            -location $tx_port_location]
```

In this example, the call to `create` sets the port location attributes to override the default values. (You can also call the `config` function to set attributes after you have created an object.)

- During test execution, you can retrieve the value of result attributes. Then you can change the test during execution by using the `config` function to modify attributes for the test configuration objects. You can also modify object attributes after the test has finished, and run the test again, without having to recreate the object hierarchy.

When you call the `config` function, you specify the object handle and one or more attribute name-value pairs. Spirent TestCenter modifies the values of the attributes as specified in the call.

In the previous example, the port location was specified in the create function call. You can also set the location by calling the `config` function:

```
stc::config $tx_port -location $tx_port_location
```

In this case, the call to the `config` function specifies a Port object handle (`tx_port`), the name of the attribute (`-location`), and the value for the attribute.

Retrieving Attribute Values

To retrieve the value of an object attribute, use the **get** function. When you call **get**, you specify the handle for an object and, optionally, the name of one or more of its attributes; the function returns the value of one or more attributes. In the following example, the **get** function returns the values for all of the attributes for the **Port** object.

```
set txPortAttr [stc::get $tx_port]
```

In the following example, the **get** function returns the value of the location attribute:

```
set txPortLocation [stc::get $tx_port -location]
```

Descendant-Attribute Notation

Spirent TestCenter Automation syntax supports a path notation called Descendant-Attribute Notation (DAN). An attribute specification that uses DAN includes an object path (object type names separated by periods). For example, the following call to the **config** function modifies the **–active** attribute for the first **Port** child of the **\$project** object.

```
stc::config $project -Port.active false
```

DAN specification allows you to reference attributes of descendant objects without retrieving additional object handles. For a complete description of Descendant-Attribute Notation, see the *Spirent TestCenter Automation Programmer's Guide*.

Automatic Object Creation

In certain cases, Spirent TestCenter creates objects on your behalf. In these situations, when you create an object, Spirent TestCenter creates the object that you specify in the function call, and it creates one or more additional objects.

For example, when you create a **StreamBlock** object, Spirent TestCenter also creates Ethernet and IPv4 header objects. The following example shows the function calls to create a **StreamBlock** object, and then retrieve the handles to the automatically created and **EthernetII** and **IPv4** objects.

```
set streamBlock [stc::create streamBlock -under $tx_port]
set ethhead \
  [stc::get $streamBlock -children-ethernet:EthernetII]
set ip4head [stc::get $streamBlock -children-ipv4:Ipv4]
```

Note that the example uses the following syntax to retrieve a child object:

```
-children-ethernet:EthernetII
```

This child specification uses the **ParentChild** relation between the stream block and its children. The specification consists of an abbreviated relation reference (**–children**) followed by an object type name (**-ethernet:EthernetII**). The abbreviated reference is called a *side name*. By specifying an object type (**ethernet:EthernetII**), Spirent TestCenter

filters the results of the retrieval operation, returning only the handles for the **EthernetII** child objects of the specified **StreamBlock** object. For more information about relations, see the following section.

Using Relations

Spirent TestCenter uses relations to manage the connections between objects in your test configuration. The most common type of relation is the **ParentChild** relation. When you create an object, Spirent TestCenter automatically creates the **ParentChild** relation between the newly created object and the object identified by the **-under** parameter in the call to the **create** function. For example:

```
set tx_port [stc::create Port -under $project] \  
            -location $tx_port_location]
```

For those objects that Spirent TestCenter creates automatically, it also creates the corresponding **ParentChild** relations. For example, when you create a **Port** object, Spirent TestCenter also creates **Analyzer** and **Generator** objects automatically, and it creates the **ParentChild** relations to connect the objects to the **Port** parent.

You can retrieve the handles for the children of an object by specifying the **-children** side name in a call to the **get** function. A *side name* is a single name that corresponds to a relation type and a specific side of that relation. For example, the following call returns a list of all of the children of the **tx_port** object:

```
stc::get $tx_port -children
```

You can filter the resulting list by appending a specific type to the side name. The following function call returns the handle to the **Generator** child of the **tx_port** object.

```
set generator [stc::get $tx_port -children-generator]
```

Note that Spirent TestCenter supports filtering of child objects only. For more detail about side names, see the information about syntax in the Spirent TestCenter *Automation Programmer's Guide*.

In addition to **ParentChild** relations, Spirent TestCenter defines other relation types. In some cases, you must create relations to support specific test operations. Use the **config** function to create relations. When you call **config**, you specify an object and a relation reference. The relation reference is a side name that specifies the relation type and the remote side of the relation (source or target), along with the handle to the object to be used for the remote side of the relation.

The following example shows how to create an **AffiliationPort** relation:

```
stc::config $txHost -AffiliatedPort $txPort
```

In this case, the **AffiliationPort** relation represents a connection between **Host** and **Port** objects. The call to **config** specifies the **-AffiliatedPort** side name for the **txHost** object. The side name corresponds to the target side of the **AffiliationPort** relation; the relation reference identifies the remote object in the relation (**txPort**).

Test Output

When you run a Spirent TestCenter test, Spirent TestCenter Automation produces two types of output:

- *Results data* include all test results. Spirent TestCenter maintains test results in memory by setting the value of result attributes for objects in the object hierarchy. During a test, you can obtain test results by using the **get** function to retrieve the value of result attributes. After the test has completed, Spirent TestCenter produces output files containing the end-of-test results data. Spirent TestCenter Automation also produces a log file.
- *Capture data* are network traffic (frame data) that has been collected according to a specified filter.

The following sections describe how to use the API to access test results and how to use Spirent TestCenter Automation to obtain capture data.

- [Spirent TestCenter Test Results](#)
- [Data Capture](#).

Spirent TestCenter Test Results

Spirent TestCenter test results are collected for the duration of a test and, at the end of a test run, Spirent TestCenter Automation writes the test results to files. To enable the collection of test results, you must establish subscriptions for results.

The following sections provide information about:

- [Log Files: The Automation Options Object](#)
- [Result Files](#)
- [Using the API to Retrieve Test Results During Test Execution](#)
- [Using the API to Retrieve Test Results from a Results Database](#).

Log Files: The Automation Options Object

Spirent TestCenter defines the **AutomationOptions** object to store settings for logging and Tcl error handling. The object has the following attributes:

- **LogLevel** - Defines the minimum severity level of logged diagnostic messages.
- **LogTo** - Specifies the output destination for diagnostic messages.
- **SuppressTclErrors** - Indicates whether or not Tcl errors will be suppressed.

Spirent TestCenter creates the **AutomationOptions** object automatically. To set automation options, first retrieve the handle for the object, and then call the **config** function to set the values. For example:

```
set hOptions [stc::get system1 -children-AutomationOptions]
stc::config $hOptions -LogLevel ERROR
```

The call to **config** specifies that Spirent TestCenter will report ERROR level messages only.

Result Files

Spirent TestCenter produces two kinds of results files:

- Files formatted with comma-separated values (.csv file extension) – These files are suitable for display using a spreadsheet program, for example, Microsoft Excel, OpenOffice Calc, or Gnumeric.
- A end-of-test results database file – You use Spirent TestCenter Automation commands to create the database and to retrieve results from the database. You can also use the Results Reporter to view the contents of the database. (The Results Reporter is a Spirent TestCenter GUI application.)

In order to generate results files, you must use the **subscribe** function. To generate .csv files, specify an output file when you establish a subscription. To generate a results database, use the **SaveResult** command.

When you call the **subscribe** function, you identify the type of results to be collected, and you specify the set of objects for which the results will be collected. By default, Spirent TestCenter Automation does not produce a result output file. To produce an output file you must specify the **-filenamePrefix** parameter. The following example shows a subscription for port results collected from the analyzer. The results will be written to the file `APR_results.csv` in the current default directory.

```
set rx_resinfo [stc::subscribe -parent $project \
                        -resultParent $rx_port \
                        -configType analyzer \
                        -filenamePrefix "APR_results" \
                        -resultType analyzerPortResults]
```

For more information, see the description of the **subscribe** function in the *Spirent TestCenter Automation Programmer's Guide*.

Using the API to Retrieve Test Results During Test Execution

When you run a Spirent TestCenter test, Spirent TestCenter Automation maintains a copy of the test results by setting the value of result attributes defined for the objects in the test object hierarchy. You can retrieve test results at any time during test execution, depending on when a particular type of test result is available.

- Spirent TestCenter Automation continuously updates result attributes that reflect the value of real-time counters or any discrete test measurement taken during test execution. You can retrieve these values at any time.
- Spirent TestCenter Automation derives certain test results as statistics that reflect data collection over time. This type of test result is available at the end of the test run.

To retrieve test result values, use the **get** function. The following example shows a call to the **get** function that retrieves the value of the transmission frame count for the entire test configuration.

```
set frame_count [stc::get txFrameCnt -from $project]
```



Note: You can retrieve test results during test execution, and you can modify the attributes of your test configuration in response to those results. When you modify attributes during a test, you must call the **apply** function to activate the changes. When you call the **apply** function, Spirent TestCenter Automation sends the modifications to the chassis. The executing test is modified at that point, and the result values will reflect the changes in real time. Depending on your configuration, the type of test you are running, and the type of modifications to the configuration, there may be a short period of time during which your results are inconsistent.

Using the API to Retrieve Test Results from a Results Database

You can use Spirent TestCenter Automation to save end-of-test results in a database and to retrieve results from the database.

- To create an end-of-test results database, use the `SaveResults` command.
- To retrieve results from an end-of-test results database, use the `QueryResult` command.

For more information about creating and accessing a results database, see the *Spirent TestCenter Automation Programmer's Guide*.

Data Capture

You can direct Spirent TestCenter Automation to capture frame data during a test. Spirent TestCenter performs the capture operation at the port level. To support capture operations, Spirent TestCenter automatically creates a set of capture objects for each **Port** object. To start capture for a port, you must retrieve the handle of the **Capture** object associated with the port, and use the handle with the **CaptureStart** command. For more information about using the Spirent TestCenter capture support, see the *Spirent TestCenter Automation Programmer's Guide*.

Spirent TestCenter Automation Run-Time Environment

Spirent TestCenter Automation supports the following Tcl software:

- Tcl Version 8.3.5 and 8.4.5

Creating Test Configurations

Figure 3-1 shows an object hierarchy for a simple test that generates traffic and collects results data. (This is the object hierarchy shown in *Figure 2-4 on page 21*.) This test configuration is used to generate traffic containing Ethernet and IPv4 headers. The figure also shows the addresses used for the example.

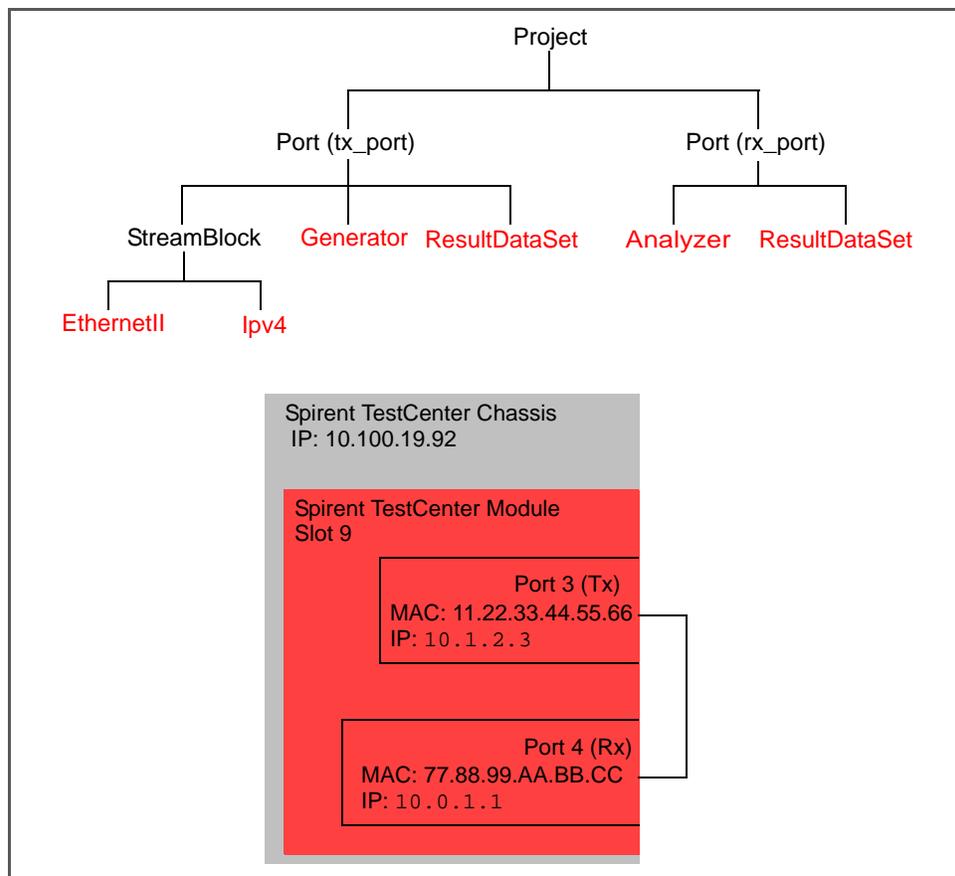


Figure 3-1. Object Hierarchy for a Spirent TestCenter Frame Loss Test

The following sections provide an example of creating and using this object hierarchy. The syntax of the Spirent TestCenter API gives you the flexibility to set attribute values when you create an object or at some point afterward. The example shows both methods. Note that parents are created before children. This is necessary because the parent object handle is a required parameter in the call to create a child object.

The following list shows the general steps to create a test configuration and run the test:

- 1 Initialize the interface.
- 2 Connect to the Spirent TestCenter chassis and reserve the ports that you intend to use.
- 3 Create the object hierarchy. The basic set of objects includes the project object (the root of the hierarchy), port, router, and traffic-related objects. Choose the specific set of object types that is appropriate for the test type. (For a complete list of object types, see the *Spirent TestCenter Automation Object Reference*.)
- 4 Establish a subscription for test results.
- 5 Start the test.
- 6 After the test is complete, release the ports and disconnect from the chassis.

The following sections describe an example that shows these steps.

Using the Tcl Interface (An Example)

The following sections describe a short Tcl script that creates a test configuration for a test that generates traffic and collects results.

- [“Initialization”](#)
- [“Chassis Connection and Port Reservation” on page 37](#)
- [“Project and Port Objects” on page 37](#)
- [“Traffic Configuration” on page 38](#)
- [“Result Subscription” on page 39](#)
- [“Test Execution” on page 41](#)
- [“Cleanup” on page 42.](#)



Note: You can create a test configuration either by using a file containing the Tcl statements, or you can use a Tcl shell window to enter the statements one at a time.

Initialization

At the beginning of your test session, you must initialize the Spirent TestCenter Automation interface. Spirent TestCenter Automation supplies the Tcl package **SpirentTestCenter** that defines the Spirent TestCenter API. To use the API, include the following line at the beginning of your Tcl script:

```
# required
package require SpirentTestCenter
```

Chassis Connection and Port Reservation

To use a Spirent TestCenter chassis for a test, you must connect to the chassis and you must also reserve the ports that you intend to use on the chassis.

The following code fragment shows the calls to connect the chassis and reserve the ports.

```
# Define variables for the chassis IP address and the ports.
# Assume that both ports are on the same chassis.

set ip 10.100.19.92
set tx_port_location $ip/9/3
set rx_port_location $ip/9/4

# Connect to the chassis and reserve the ports.
stc::connect $ip
stc::reserve $tx_port_location
stc::reserve $rx_port_location
```

- To connect to a Spirent TestCenter chassis, specify the IP address of the chassis.
- When you call the **reserve** function, you specify the chassis IP address, together with the slot number and the port number. Note that chassis-slot-port specification identifies the group to which the port belongs. In this example, ports 3 and 4 belong to the same group. It is necessary to explicitly reserve both ports so that Spirent TestCenter will perform the appropriate initialization for each port. (For more information about port groups, see the description of the **reserve** function in the *Spirent TestCenter Automation Programmer's Guide*.)



- Notes:**
- When you reserve a port, you establish exclusive access to the port.
 - Port numbering begins at 1.

Project and Port Objects

The first object that you create must be a **Project** object that will serve as the root of the test hierarchy. Once you have created a **Project** object, you can then create the objects that will describe the components of your test configuration.

A **Port** object is a child of the **Project** object. A **Port** object identifies the location of a port that you will use on the Spirent TestCenter chassis. **Port** objects are a logical representation of physical ports on a Spirent TestCenter chassis. Note that after you create the **Port** objects for your test, you must execute the **SetupPortMappings** command to establish the mapping between the logical and physical ports.

When you create a **StreamBlock** object, Spirent TestCenter automatically creates **EthernetII** and **Ipv4** header objects. As an alternative, you can specify a different format using the **FrameConfig** attribute of the **StreamBlock** object.

This code fragment shows Tcl statements that create a **Project** and two **Port** objects, and establish the port mappings. This example shows both methods of setting attributes – either by specifying attribute name-value pairs in the call to the **create** function or by using the **config** function.

```
# Create the Project object
set project [stc::create project]

# Create the first port.
set tx_port [stc::create port -under $project]

# Set the physical location of the port, identifying the chassis, slot, and port
# (often referred to as "c/s/p").
stc::config $tx_port -location $tx_port_location

# Create the second port and set the location at the same time
set rx_port [stc::create port -under $project -location $rx_port_location]

# So far we have created logical ports and we have reserved physical
# ports. Now we must associate each logical port with a physical port.
# This depends on the port locations just established.
stc::perform setupPortMappings
```



Note: Remember that in order to use a port, you have to create the **Port** object and you must reserve and map the port. (For an example of reserving a port, see [“Chassis Connection and Port Reservation” on page 37.](#))

Traffic Configuration

StreamBlock and frame header objects define the traffic streams that Spirent TestCenter will generate from a port. A **StreamBlock** object is a child of a **Port** object. You can create multiple **StreamBlock** objects for a single **Port** object.

When you create a **StreamBlock** object, Spirent TestCenter automatically creates **EthernetII** and **Ipv4** header objects. Use the protocol header objects to define the source and destination addresses for the communication generated from the ports. To set the header object attributes, you must first retrieve the header object handles from the **StreamBlock** object.

The following code fragment shows the function calls to configure traffic for the example. This code fragment:

- Creates a **StreamBlock** as a child of the transmitting **Port** object
- Retrieves handles for the automatically created **EthernetII** and **Ipv4** objects
- Sets the source and destination MAC addresses in the **EthernetII** object

- Sets the source and destination IP addresses in the **Ipv4** object
- Calls the **apply** function to send the configuration to the chassis.

```
# Generated traffic comes from stream blocks. There can be multiple
# stream blocks associated with each port. For this example we will
# have just one. Note that stream blocks are created under ports.
set streamBlock [stc::create streamBlock -under $tx_port]

# By default, a stream block is automatically configured with
# EthernetII and IPv4 headers. Get their handles.
set ethhead [stc::get $streamBlock -children-ethernet:EthernetII]
set ip4head [stc::get $streamBlock -children-ipv4:Ipv4]

# Set the MAC addresses in the the ethernet header.
stc::config $ethhead -srcMac "11.22.33.44.55.66"
stc::config $ethhead -dstMac "77.88.99.AA.BB.CC"

# Set the IP addresses too. You can configure multiple attributes
# at once.
stc::config $ip4head -sourceAddr "10.1.2.3" -destAddr "10.0.1.1"

# Send the configuration to the chassis.
stc::apply
```

Result Subscription

To obtain results, you establish subscriptions for sets of result attributes. The following code fragment establishes subscriptions for both the generator and the analyzer. The code fragment:

- Retrieves the generator object handle
- Establishes a subscription for **GeneratorPortResults** data. The call to the subscribe function specifies:
 - The **Project** object handle
 - The object handle of the parent for the automatically created results objects (-resultParent)
 - An object type that indicates the set of results (-resultType)
 - The object type of the source object in the **ResultChild** relationship with the configuration type object (-configType). In this case, the **Generator** object is the source object and the **generatorPortResults** object is the target.
- Executes the same calls (handle retrieval, subscription) for the analyzer
- Suspends script execution for 3 seconds to allow time for configuring the subscriptions
- Retrieves handles to the result objects for later use in retrieving result attribute values.

```
# Every port comes ready equipped with a packet generator, a packet
# analyzer, and lots of other things too. For now, just get the handle
# of the packet generator for the transmit port.

set generator [stc::get $tx_port -children-generator]

# Set up to get statistics from the generator. tx_resinfo becomes
# the handle of the resultDataSet object, which is mostly a list
# of the individual result objects. For this example we only subscribe
# to a single set of results.

set tx_resinfo [stc::subscribe -parent $project          \
                             -resultParent $tx_port    \
                             -configType generator     \
                             -resultType generatorPortResults]

# Same thing on the receive side for the analyzer
set analyzer [stc::get $rx_port -children-analyzer]
set rx_resinfo [stc::subscribe -parent $project          \
                              -resultParent $rx_port    \
                              -configType analyzer      \
                              -resultType analyzerPortResults]

stc::sleep 3

# From the results info objects we can get the handles of
# results objects themselves. These are the objects that will
# contain the actual results.
set tx_results [stc::get $tx_resinfo -resultHandleList]
set rx_results [stc::get $rx_resinfo -resultHandleList]
```

Figure 3-2 on page 41 shows the objects in the test configuration that are identified by the subscriptions. Based on the subscription, Spirent TestCenter will create the objects that it needs to store result values (in this example, the **ResultDataSet**, **GeneratorPortResults**, and **AnalyzerPortResults** objects).

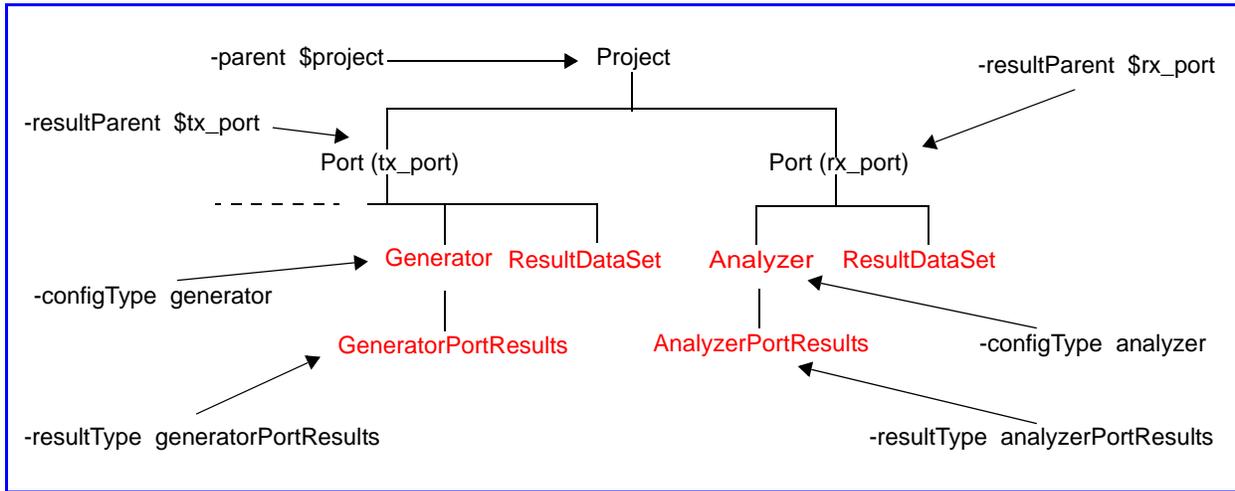


Figure 3-2. Subscription

In the figure, the automatically created objects are shown in red. Note that when you call the **subscribe** function, some attributes require handle values (**-parent**, **-resultParent**) and some attributes require object type values (**-configType**, **-resultType**). The **subscribe** function returns a handle to a **ResultDataSet** object. Spirent TestCenter uses this object to manage the result objects. You use the **ResultDataSet** handle to obtain handles to the individual result objects.

Test Execution

Once you have created and applied your test configuration, and you have established subscriptions, you can execute the test. To execute the test, start the analyzer and generator. The following code fragment uses the **analyzerStart** and **generatorStart** commands, providing the analyzer and generator object handles that were retrieved earlier.

```
# Start the traffic generation and analysis. The test run lasts
# 15 seconds. The generator is stopped after ten seconds, but
# the analyzer is allowed to continue for five more seconds
# to catch any long delayed packets.

stc::perform analyzerStart -analyzerList $analyzer
stc::perform generatorStart -generatorList $generator
```

The following code fragment contains a loop that retrieves transmit and receive counts, once every second. The generator is stopped after 10 seconds, but the loop continues for another five iterations to account for any delayed packets.

```
puts [format "%2s %12s %12s %12s %12s" \
           "T" "Tx frames" "Tx octets" "Rx frames" "Rx octets"]
for {set t 0} {$t < 15} {incr t} {
    array set rx_counts [stc::get $rx_results]
    array set tx_counts [stc::get $tx_results]
    puts [format "%2d: %12d %12d %12d %12d" $t \
               $tx_counts(-GeneratorFrameCount) \
               $tx_counts(-GeneratorOctetCount) \
               $rx_counts(-TotalFrameCount) \
               $rx_counts(-TotalOctetCount) ]
    if {$t == 10} {
        stc::perform generatorStop -generatorList $generator
    }
    stc::sleep 1
}
stc::perform analyzerStop -analyzerList $analyzer
```

Cleanup

After the test has completed, your script should do the following:

- Release any reserved port groups. You must release all the port groups that you have reserved. Note that a port group may contain more than one port; any attempt to use a port handle after it has been released will result in an error. (For a description of the **release** function, see the *Spirent TestCenter Automation Programmer's Guide*.) An alternative to releasing ports is to exit the Tcl shell. This will also unreserve any ports, disconnect from the chassis' and release any resources.
- Disconnect from the chassis.
- Delete the **Project** object to release any resources accumulated through use of the API.
- Reset the configuration to return Spirent TestCenter Automation to its initial state.

The following code fragment shows an example of the function calls that you use to perform cleanup.

```
# Clean up.
stc::release $tx_port_location
stc::release $rx_port_location
stc::disconnect $ip
stc::delete $project
stc::perform resetConfig
```

Test Completion

In some situations, after you have created your test configuration and started test execution, you may wish to suspend execution of your application until the sequence of commands has completed. Spirent TestCenter Automation provides the **waitUntilComplete** function for this purpose. The **waitUntilComplete** function is a blocking function. When you call the function, Spirent TestCenter Automation does not return control to your application until the sequence has completed.



Appendix A

Spirent TestCenter Licenses

In order to use Spirent TestCenter Automation, you must have the correct Spirent TestCenter license(s). When you purchase a Spirent TestCenter system, the system box (containing the hardware, software, and installation documentation) contains a License Authorization Code (LAC).

- Use the License Authorization Code to obtain the license key files (LIC) for the software packages that you have purchased. You obtain license key files from the Spirent Communications Customer Support Center (CSC) web site. For more information about this process, see the *Getting Started with Spirent TestCenter document* that was included with your Spirent TestCenter system.
- License key files are saved in a directory on your PC. Use the Spirent TestCenter application to install licenses on the chassis controller. For information about how to install licenses, see the Spirent TestCenter online help.



Index

A

- API functions 25
- API syntax 26
- API, Spirent TestCenter 13
- apply function 34
- Attributes, object 29
- AutomationOptions object 32

C

- Capture data 32
- Chassis, Spirent TestCenter 16
- Common object types 18
- create function 28
- Creating a Spirent TestCenter test 20
- Creating object handles 28
- Creating test configurations 35

D

- DAN 30
- Data capture 34
- DDN 29
- Descendant-Attribute Notation 30
- Direct-Descendant Notation 29

E

- End-of-test results database 33

F

- Functions 25

G

- get 28, 33

H

- Handle, object. See Object handle

L

- Log files 32

N

- Network test environment 16

O

- Object attributes 29
- Object handle 28

- creating 28
- using 28
- Object hierarchy 15
 - duration 18
 - organization 21
 - Tcl example 43
- Object types 18

P

- Path notation 29, 30
- PC connection to chassis 16
- perform function 28

R

- related documentation 6
- Relations 31
- Results data 32
- Results database 33
- Results file (.csv format) 33
- Root of object hierarchy 21

S

- Software model (object hierarchy) 16
- Spirent TestCenter
 - benefits 11
- Spirent TestCenter API 13
- Spirent TestCenter chassis 16
- Spirent TestCenter Conformance 11
- Spirent TestCenter fixed duration test
 - figure 21
- Spirent TestCenter test, creating 20
- subscribe 33
- Syntax, API 26

T

- Tcl error handling 32
- Tcl Example
 - creating object hierarchy 43
- Tcl version 35
- test
 - general steps to set up 20
- Test configuration, creating 35
- Test environment, network 16
- Test object
 - hierarchy root 21
- Test output 32

Test results
 retrieving 33
Test steps 20
Types, object 18

U

Using object handles 28