OmniBER XM network simulator

API Programming Guide

Agilent Technologies

Agilent OmniBER XM Network Simulator

API Programming Guide

Agilent Technologies

Notices

© Agilent Technologies, Inc. 2001

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Sales and Service Offices

An up-to-date list of Agilent Offices is available through the Agilent Website at URL: http//www.agilent.com

Manual Part Number

J7241-90012

Edition

First edition, February 2004

Printed in UK

Agilent Technologies UK Limited

South Queensferry, West Lothian, Scotland

EH30 9TG

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In this Programming Guide

You will find information on how to control the OmniBER XM programmatically, through the application programming interface (API).

1 Introduction to the API

This chapter explain the system API and the choice of Syntax available (SCPI or Tcl), and also the stages involved in running a typical test session.

2 Example Session

This chapter gives step by step instructions on how to set up and run an example test session. At each stage the API commands used are explained.

3 Tcl Shell-Interactive Control

This chapter explains how the Tcl Shell can be used to input any API command from a command prompt.

4 Command Reference

This chapter lists and gives explanations of all the commands necessary to operate the OmniBER XM remotely.

5 Objects

Provides a list of OmniBER XM objects.

Copyright

© Copyright Agilent Technologies, 1999 — 2004.
All rights reserved.

Notice

The information contained in this document is subject to change without notice.

AGILENT TECHNOLOGIES MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Trademarks

Agilent Technologies, the Agilent logo and OmniBER XM network simulator are trademarks of Agilent Technologies.

Microsoft®, MS-DOS®, Windows®, Windows NT®, and Windows 2000® are U.S. registered trademarks of Microsoft Corporation.

Adobe®, Acrobat®, and the Acrobat Logo® are trademarks of Adobe Systems Incorporated.

UNIX® is a registered trademark of the Open Group.

Warranty and Product support

A copy of the specific warranty terms applicable to your product and replacement parts can be obtained from your local Agilent Technologies representative. Should you require technical assistance, contact your local Agilent Technologies representative. For contact information, refer to the OmniBER XM network simulator User Guide.

Printing history

New editions of this guide are issued to reflect extensive changes made to the product. Revisions may be issued, between editions, to correct errors in the manual.

Manual Name: OmniBER XM network simulator API Programming Guide

Edition	Publication Date
1.0	February 2004

Printed in U.K.

Certification

Agilent Technologies certifies that this product met its published specifications at the time of shipment from the factory.

OmniBER XM network simulator: Agilent Technologies further certifies that calibration measurements made on its manufactured network simulator interface modules are traceable to the United States National Institute of Standards and Technology, to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

Additional Information for Test and Measurement Equipment

To comply with EMC regulations, shielded cables should be used on all appropriate connections. Otherwise, the user has to ensure that, under operating conditions, the Radio Interference Limits are still met at the border of the user's premises.

Warnings

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Agilent Technologies assumes no liability for the customer's failure to comply with these requirements.

Ground the Equipment: For safety, Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety ground must be provided from the mains power source to the product input wiring terminals or supplied power cable. Before operating the equipment, guard against electric shock in case of fault by always using the provided 3-conductor power cord to connect the equipment to a grounded power outlet.

DO NOT use in hazardous environments: Do not operate the product in an explosive atmosphere or in the presence of flammable gases or fumes. This product is designed for indoor use only.

DO NOT use repaired fuses or short-circuited fuse holders: For continued protection against fire, replace line fuses only with fuses of the same voltage and current rating and type.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers and shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to an Agilent Technologies Sales and Service Office for service and repair to ensure the safety features are maintained.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to an Agilent Technologies Sales and Service Office for service and repair to ensure features are maintained.

DO NOT clean with fluids: Doing so may make the equipment unsafe for use. Power down the equipment and disconnect the power cord before cleaning. To clean, use a soft dry cloth.

Safety Symbols

If you see this symbol on a product, you must refer to the manuals for specific Warning or Caution information to avoid personal injury or damage to the product.

Indicates the field wiring terminal that must be connected to ground before operating the equipment. Protects against electrical shock in case of fault.

or

Frame or chassis ground terminal. Typically connects to the equipment's metal frame.

Alternating current (ac).

Direct current (dc).

Indicates hazardous voltages and potential for electrical shock.

Indicates that antistatic precautions should be taken.

Indicates laser radiation when turned on.

This product complies with CSA requirement CSA 22.2 No. 1010.1, NRTL/C, EN 61010-1:1993 + A2:1995/IEC 1010-1:1990 + A1:1992 + A2:1995 Safety requirements for electrical equipment for measurement, control, and laboratory use.

Notice for European Community: This product complies with the relevant European legal Directives: EMC Directive 89/336/EEC and Low Voltage Directive 73/23/EEC.

Das CE-Zeichen zeigt die Übereinstimmung mit allen für das Produkt geltenden Direktiven der Europäischen Union an.

ISM 1–A

This is the symbol for an Industrial, Scientific, and Medical Group 1 Class A product.

Dieses Zeichen steht für ein Produkt der Gruppe 1, Klasse A, für den Einsatz im industriuellen,

wissenschaftlichen und medizinischen Bereich.

This product meets the requirements of the Australian EMC Framework (AS/NZS 2064.1/2 for ISM:1A), enforced by the Radiocommunications Act 1992.

WARNING

Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION

Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

Contents

Index

1
Introduction to the API

Overview

You can control the network simulator programmatically, through the application programming interface (API). This allows you to automate tests, for example, to:

- Run tests that would be too tedious or imprecise to do manually or repeatedly through the Graphical User Interface (GUI)
- Integrate tests with larger test suites that access other test equipment and Systems Under Test
- Repeat tests for subsequent product builds
- Regression test new versions or releases of products

The network simulator is easily integrated into existing or new production test applications using the API. The text/sockets-based commands are implemented in any programming environment quickly and simply. With a few commands, development engineers can build applications to:

- Connect to the network simulator
- Configure ports
- Configure a test
- Run tests
- Collect results
- Clear down

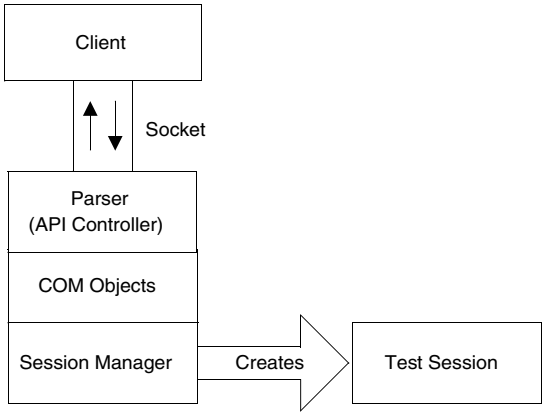
Tcl syntax

AgtInvoke <object> <function> [parameters…]

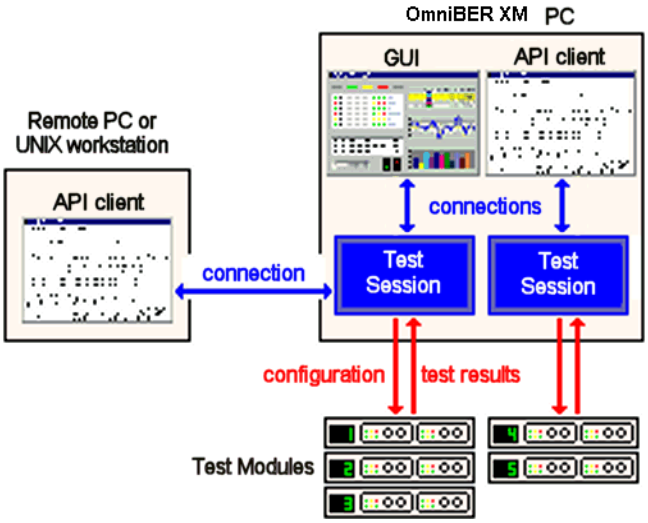
Example AgtInvoke AgtPortSelector AddPort <modules> <port>

Sessions

The API is multi-user. Clients can create new sessions or join existing ones. The Session manager controls and allocates sessions.



Clients may communicate with the tester locally, from the same PC, or remotely from another PC or UNIX workstation. The example given in the following page shows two typical configurations.

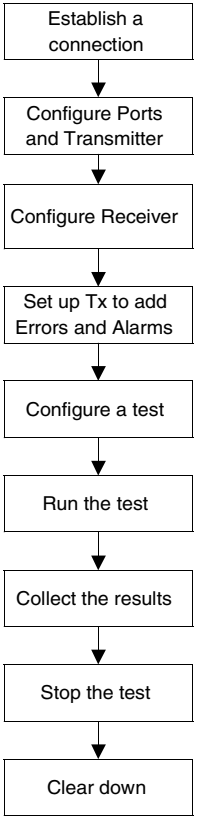


Clients communicate with the tester through an Agilent-supplied package of commands. These commands send and receive messages through the general-purpose, line-oriented, TCP socket connection. Through this connection, API clients have full access to the tester's capabilities. Changes made through the API are reflected instantly in the GUI.

A *test session* is simply an instantiation of the tester software on the host PC. A session reserves contiguous blocks of *test modules* for its exclusive use, and processes commands from its API and GUI clients. You can run multiple sessions concurrently, and have multiple API and GUI clients connected to each session. Thus, you can configure and start each test session independently; the test ports make the actual real-time measurements and pass results back to the test sessions.

Session Stages

The stages in running a typical session are as follows:



1. A connection is made to the session manager and a port to talk to the session on is established.
2. You add the ports to the session, turn on the port lasers and configure the Transmitter. This may include selecting Terminal or Thru mode operation, Signal Mode, Optical Wavelength, Clock Source, Channels and Payload Pattern.
3. Then you set up the Receiver Signal Mode, Channels and Pattern.
4. Set up the Transmitter to add Errors and Alarms.
5. This stage might involve setting the duration of the test, the sampling interval, or the test mode.
6. Then you issue the command to run the test.
7. Results are collected as the test is running and can be displayed or logged to a file.
8. The test is stopped.
9. Finally the lasers are turned off and the session closed.

1 Introduction to the API

2
QuickTest

About QuickTest

QuickTest is a comprehensive set of test scripts and tools that automate and simplify the testing of devices using XM. The QuickTest package also includes a script collection browser, a set of layered libraries, and a code generator which enable you to develop automated tests to meet your specific testing needs.

Creating your own QuickTest scripts

You can develop your own test scripts quickly and easily using QuickTest by accessing the standard template located on the browser's toolbar. Alternatively, if you would like to add to or change the features of an existing QuickTest, click the Clone button on the toolbar and use the instructions below to guide you through the rest of the process. See below for QuickTest Documentation including API references.

To create a new script

- In the QuickTest GUI, click the Create icon on the toolbar to create a new script using the standard template.
- Select a script type. A test automates a complete test scenario. A tool automates part of a test scenario. Now select a name for this script.
- Your script will be stored in the folder:
  C:\Program Files\Agilent\OmniBERXM\QuickTest\ UserScripts\ UserScripts\<scriptname>
  (or the target directory you used to install XM) and will be accessible from the browser.

To edit a script

Each script file contains the following files:

<scriptname>.app.tcl
The Application Library file contains the bulk of the test's code. You can use the functions available in AgtTsuLib.tcl, along with the standard XM Tcl API, as the basis for your code.

<scriptname>.cfg.tcl
The Configuration file is automatically generated by the GUI script for consumption by the Application script. You can create copies of this file with different sets of parameter values for multiple test iterations.

<scriptname>.ini.tcl
The Initialization file contains parameter default values and initialization.

<scriptname>.description.txt
The Description file contains a concise overview of the test scenario or tool. You can use existing descriptions as guides to creating a description of your test or tool.

<scriptname>.diagram.gif
The Diagram file contains a diagram that summarizes the test scenario. This graphic should provide a good overview of the test and should complement the text description described above. Save your diagram as a .gif file called <scriptname>.diagram.gif

<scriptname>.gui.tcl
The GUI application file contains the code that is used to display the GUI. You can find useful functions for editing the GUI in AgtTsuLib.tcl, AgtSgaLib.tcl, and AgtQuickTestLib.tcl.

<scriptname>.help.txt
The Online Help file contains a list of the steps involved in configuring and running a script.

QuickTest documentation

Reference documentation for the AgtTsu library

Reference documentation for the AgtSga library

Reference documentation for the AgtQuickTest library

QuickTest Tutorial (Microsoft PowerPoint Presentation)

3
Example session

Introduction

In this chapter we describe each stage in an example test session. At each stage, the API commands are explained.

Error and Return Value Handling

Any program using the API has to cater for handling errors from the controller and receiving data in response to a command. When developing programs using this API bear in mind the following points:

1. Some commands return values in response to an action. They always return a status result, e.g. *S_OK*.
2. If the controller experiences an error when attempting to execute a command it will return an error.

Format of a Tcl reply message

[Return Value] 2 bytes 0= success Always exists	[<space>] 1 byte Always exists	[Further data Variable length Variable meaning	[OXOD] [OXOA] Fixed ending to all messages. OXOD = \r OXOA = \r

All commands sent to OmniBER XM must also be OXOD, OXOA terminated.

Step 1: Establish a connection

A client can establish two types of connection:

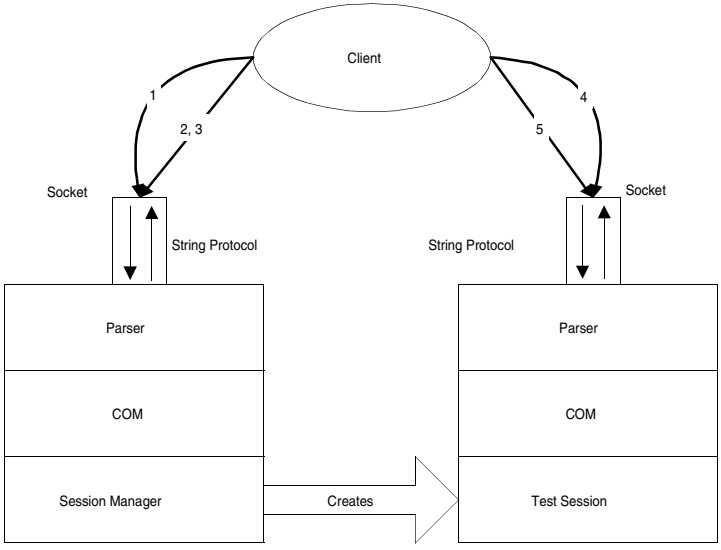
- A new session or
- An existing session.

Connect to a new test session

To launch a new test session, the client must request a test session from the Session Manager. The Session Manager then provides a TCP socket for that session through which all communication between the client and the test session must pass. This scenario describes the sequence of steps that a client will follow to launch a new test session and connect to it using the Session Manager.

Assumptions:

- Session Manager is running.
- Session Manager has a component listening for socket connections on a well known port number (i.e. 9001).



Steps:

1. A connection has to be made to the session manager on port 9001. The client opens a socket and a means of communication with the tester.
2. The OpenSession command is sent to the session manager, and then you need to call GetSessionPort to find the port number to talk to the session on. Client issues the command:

```
> AgtInvoke AgtSessionManager OpenSession OmniberXm
```

   Tester returns

```
<SessionHandle>
```

3. Client issues command to get the port number for the test session:

```
> AgtInvoke AgtSessionManager GetSessionPort <SessionHandle>
```

   Tester returns:

```
<TcpPortNumber>
```

4. Client opens socket to test session, using <TcpPortNumber>
5. Client interacts with test session by sending the API text commands over the socket interface.

NOTE

There will be multiple instances of the scripting parser, one for the session manager, one for the test session.

At Step 2, when the test session is created it creates an instance of the parser, which is listening on a particular port number. The session must return this port number to the session manager.

At Step 3, the session manager returns an error if the test session has not yet notified the session manager of its listening port.

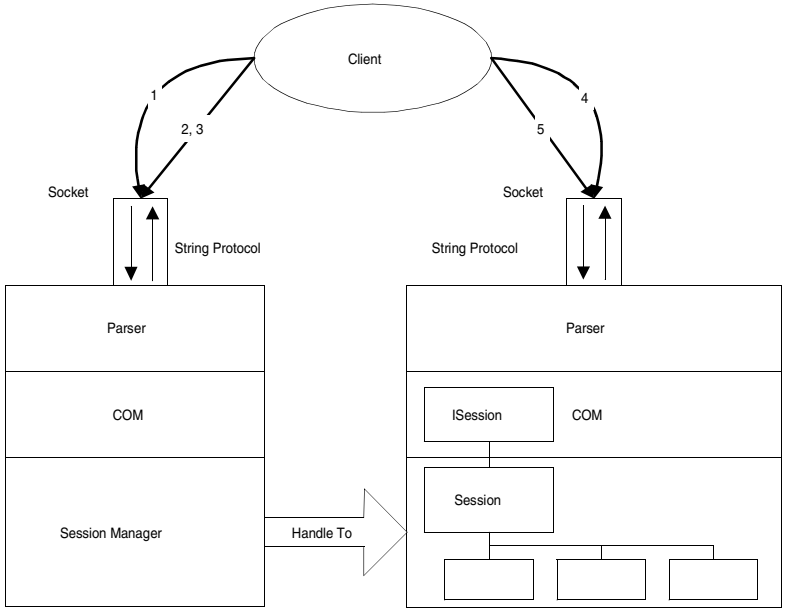
Connect to an existing test session

This scenario assumes that a session is already running and uses the services provided by the Session Manager to connect to the session.

Assumptions:

- Session Manager is running.
- Session Manager is listening for socket connections on port number 9001.
- Session Manager provides a handle to an open session.
- Test Session is listening for socket connections on a port that is registered with the Session Manager.

.



Steps

1. Client opens socket to port number 9001.
2. Client requests details of running sessions by issuing commands:

```
> AgtInvoke AgtSessionManager ListOpenSessions
```

Tester returns:

```
List<SessionHandles>
```

3 Client chooses the session of interest and issues command:

```
> AgtInvoke AgtSessionManager GetSessionPort <SessionHandle>
```

Tester returns:

```
<SessionPort>
```

4 Client opens socket to running test session.

5 Client interacts with test session by sending the API text commands over the socket interface.

Step 2: Configure the ports

At this step you configure the tester ports. You have to add the ports to the session and turn on the port lasers.

Assumptions:

- Test Session is running.
- Client has opened a socket to the Test Session.

Add tester ports

Steps: Client sends message to test session to add a port. (You need to specify a module number and port number.)

```
> AgtInvoke AgtPortSelector AddPort <module> <port>
```

Tester returns:

```
<PortHandle>
```

Turn a port laser on and confirm the action

Steps:

1 Client sends message to turn on the laser of the added port.

```
> AgtInvoke AgtOpticalInterface LaserOn <PortHandle>
```

2 Client asks for confirmation of the state of the laser.

```
> AgtInvoke AgtOpticalInterface IsLaserOn <PortHandle>
```

Tester returns:

```
<LaserState>
```

Statistics selection

Steps:

1 Client creates a statistics group to collect SONET or SDH statistics .

```
> AgtInvoke AgtStatisticsList Add AGT_STATISTICS_XM_SONET
```

Tester returns:

```
<StatisticsHandle>
```

2 Client sends message to see the list of available statistics to choose from.

```
 AgtInvoke AgtXmStatistics ListAvailableStatistics
<StatisticsHandle>
```

Tester returns:

```
<List of available statistics>
```

3 Client sends message to specify the list of statistics to be collected.

```
AgtInvoke AgtXmStatistics SelectStatistics
<StatisticsHandle> List<StatisticsEnums>
```

4 AgtInvoke AgtXmStatistics ListSelectedStatistics <StatisticsHandle>

Tester returns

```
<List of current statistics selection>
```

5 Client sends message to select a port to collect statistics from.

```
AgtInvoke AgtXmStatistics SelectPorts <StatisticsHandle>
List<ports>
```

6 AgtInvoke AgtXmStatistics ListSelectedPorts <StatisticsHandle>

Tester returns

```
<List of current ports selection for collecting statistics>
```

Step4: Configure a test

Steps:

1 Client sends message to define the test <Mode> as either,

AGT_TEST_ONCE which will run the test for the specified time or,

AGT_TEST_CONTINUOUS which will run the test until the user stops it.

```
> AgtInvoke AgtTestController SetTestMode <Mode>
```

2 If mode is AGT_TEST_ONCE client sends message to define how long (in seconds) the test will run for.

```
> AgtInvoke AgtTestController SetTestDuration <Duration>
```

3 Client sends message to find out whether or not the test is running.

```
> AgtInvoke AgtTestController GetTestState
```

Tester returns:

```
<TestState>
```

Step 5: Run the test

Steps:

1 Client sends message to start the test.

```
> AgtInvoke AgtTestController StartTest
```

2 Client requests the tester to return the status of the test.

```
> AgtInvoke AgtTestController GetTestState
```

Tester returns:

```
<TestState>
```

Step 6:Collect the results

Steps:

1 Client sends message to return the accumulated results of the selected port(s).

```
> AgtInvoke AgtXmStatistics GetAccumulatedValues
```

Tester returns:

```
<SamplingInterval> <StatisticsResults>
```

Note that the list of <StatisticsResults> will be returned when requested, irrespective of the interval of the test. The returned results will be the accumulated values of all the sampling intervals up to the time the request is issued. The <SamplingInterval> is the number of seconds that have elapsed since statistics collection started. It provides a means to order and correlate results, and to derive average statistics per interval.

Step 7: Stop the test

Steps:

1 Client may stop any test if the test <Mode> was set to AGT_TEST_CONTINUOUS (in) by sending the command.

```
> AgtInvoke AgtTestController StopTest
```

This will stop the test at any time. Another test can then be initiated or the session cleared down .

Step 8: Clear down

Steps:

1 Client sends message to turn the transmit lasers off.

```
> AgtInvoke AgtOpticalInterface AllLasersOff
```

2 Client sends message to close the session.

```
> AgtInvoke AgtSessionManager CloseSession <SessionHandle>
```

Running Example Tcl Scripts

A basic Tcl script (DemoTclScript.tcl) is provided on the CD-ROM shipped with your OmniBER XM. The Tcl script illustrates the following:

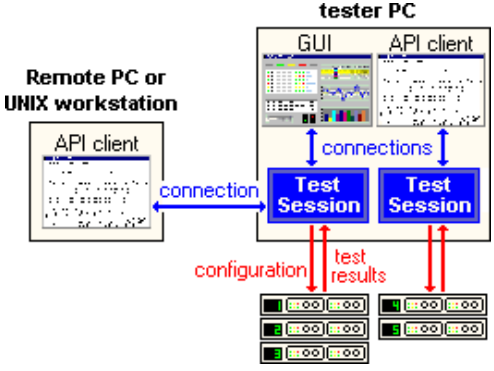
- How to create a session
- Add ports
- Gate for 20 seconds
- Add B1 errors on any port and B3 errors on all channels
- Return the B1 and B3 error count post gating

Display the number of B1 and B3 errors at the end of the gating period.

Procedure

If you are running this sample script on the XM controller (local operation) go to the **Local Operation** procedure on page 37. Steps 1 to 13 give instructions on how to set up the OmniBER XM for remote operation via a remote PC.

Remote operation via remote PC



Steps 1 to 5 explain how to install Tcl

1 If you do not already have Tcl installed you must install it now. A convenient way to do this is by using the OmniBER XM CD-ROM supplied with your system. Tcl is automatically installed when you install the OmniBER XM GUI.

2 Insert the OmniBER XM CD-ROM in your PC and once it starts click on the **OmniBER XM Software** link.

3 Select **Run this program from its current location** and then click **OK**. If you get a Security Warning dialog appearing, select **Yes**.

4 Follow the instructions on screen until you reach the **Setup Type** window, then select **Client GUI only**.

5 When the **Install extra components window** is displayed select the **Tcl** component, deselect the other component options.

Start here if you have already installed Tcl

6 On the remote computer select **Start**, then the **Run** button and in the Run dialog window type **tclsh82.**
Note you may need to change this depending on the version of Tcl you have downloaded to your machine).

7 Type **package require AgtClient**.

8 Type **AgtSetServerHostname <name of the tester XM controller>** (no <>s). Example: AgtSetServerHostname xmtest08 (where xmtest08 is the name of the tester XM Controller)

9 Type **AgtGetVersion**.

10 The sample script (DemoTclScript.tcl) is installed on your OmniBER XM CD-ROM. Insert your CD-ROM into the remote PC.

11 Select **Start**, then **Run** and enter **cmd**

12 Type the following:
D:\>tclsh82 DemoTclScript.tcl -r xmtest08 -p "101/1 102/1"
(where D is the drive containing your CD-ROM). See Note in step 6 regarding Tcl versions. Note that **-r** is followed by the name of the OmniBER XM controller (in this example xmtest08); **-p** is followed by the name of the available ports that you wish to add.

13 The program will run.

Local Operation

The OmniBER chassis and modules are controlled directly by the OmniBER XM controller; there is no remote PC involved. The required Tcl files are already installed. The basic Tcl script **“DemoTclScript.tcl”** is installed in: the OmniBER XM CD-ROM and also at:
C:\Program Files\Agilent\OmniberXM\doc\

1. Copy “DemoTclScript.tcl” into a more convenient folder (e.g.C:\Tcl\) or use the file from its current position. Make a note of the file name and saved path (example-C:\Tcl\DemoTclScript.tcl).
2. Select **Start**, then **Run** and enter **cmd**.
3. Type **D:\>tclsh82 DemoTclScript.tcl -p "101/1 102/1"** (where D is the drive containing your CD-ROM) - modify path as appropriate for your own setup. Note. you may need to change the Tcl version depending on the version of Tcl you have downloaded to your machine, it may for example be tclsh84). Note that **-p** is followed by the name of the available ports that you wish to add.
4. The program will run.

Viewing the DemoTclScript file

You can view the contents of the DemoTclScript file using Notepad.

On your PC select **Start**/**Programs**/**Accessories**/**Notepad -** browse to where the DemoTclScript file is and select **Open**.

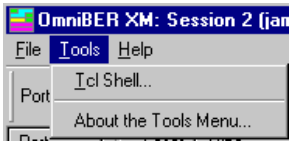
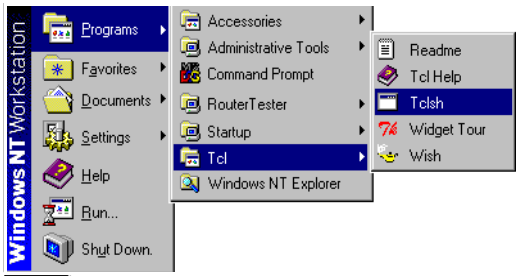
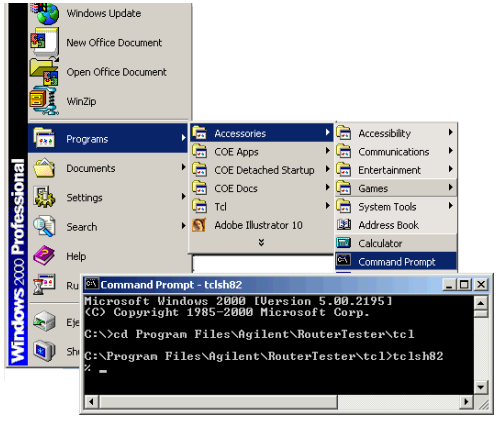
3 Example session

4
Tcl Shell - Interactive Control

Tcl Shell Overview

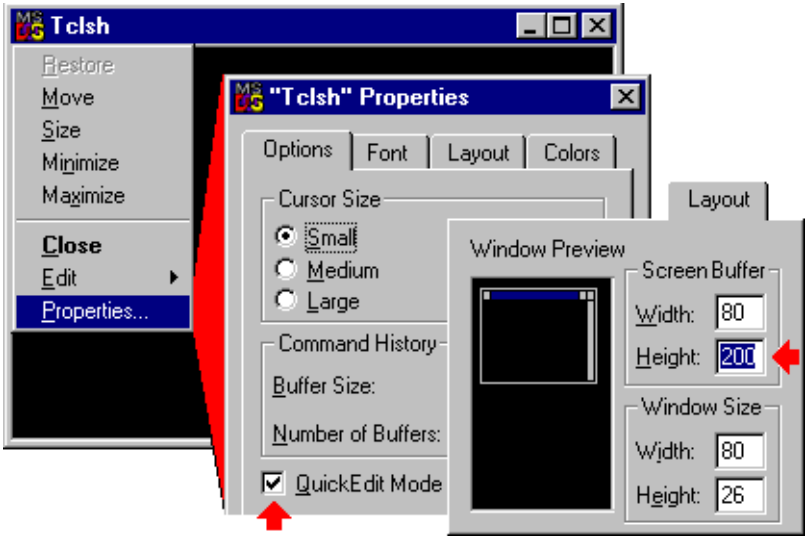
OmniBER XM network simulator is supplied with a useful interactive tool to help develop and test integration development. The Tcl shell allows input of any API command (using Tcl Syntax) from a command prompt.

To launch a Tcl shell

OmniBER XM network simulator menu		Use this method if the OmniBER XM network simulator is already running.
Windows NT Start menu		. Use this method if it is not.
DOS Command prompt		**1** Go to directory Program Files\Agilent\OmniBERXM\tcl **2** Execute **tclsh82** to start the Tcl shell.

To set Tcl shell properties

You should redefine a couple of Tcl shell properties:



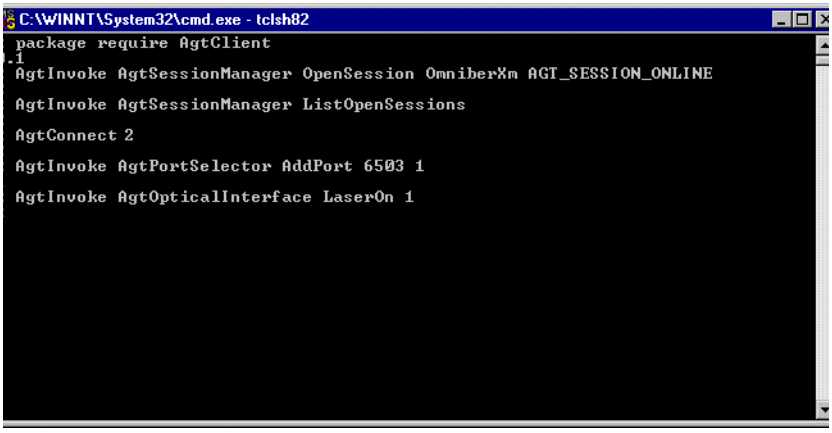
Enabling **QuickEdit Mode** lets you use the left and right mouse buttons to copy and paste text in the Tcl shell. (This is much faster than using the Edit sub-menu.)

Increasing the **Screen Buffer Height** lets you scroll the display back to see previous commands and output.

After clicking OK on the Properties dialog, select “Modify shortcut which started this window” to use the same settings every time you launch a Tcl shell.

To interact through a Tcl shell

An ideal use of the Tcl Shell is to debug integration programs. You can enter individual commands (using Tcl syntax) into a Tcl shell's command line. Again, changes are reflected instantly in the tester's GUI.



5
Commands

“Quick Reference" on page 44

Quick Reference

To communicate with the tester

`AgtConnect ?SessionHandle? -> ConnectionID`

Opens a TCP socket connection to a test session already opened by a GUI or API client

`AgtDisconnect ?ConnectionID?`

Closes a connection to a test session

`AgtListConnections -> ConnectionIDs`

Lists the connections your API client has to different test sessions

`AgtSetActiveConnection ConnectionID`

Sets the active connection, to which subsequent API commands apply

`AgtGetActiveConnection -> ConnectionID`

Returns the ID of the connection to which subsequent commands apply

To manage test sessions

`AgtListSessionTypes -> SessionTypes`

Lists the types of test sessions you may initiate

`AgtOpenSession SessionType ?SessionMode? -> SessionHandle`

Opens a new test session (that is, one not opened by a GUI) and connects to it.

`AgtCloseSession SessionHandle`

Closes a test session gracefully, stopping if there are connected clients

`AgtKillSession SessionHandle`

Closes a test session immediately, even if there are connected clients

`AgtListOpenSessions -> SessionHandles`

Lists all the currently running test sessions

```
AgtSaveSession Filename ?Objects?
AgtRestoreSession Filename ?Objects?
```

Saves a test configuration (that is, protocol settings, simulations, defined traffic) to a plain text file. Loads a previously saved test configuration.

```
AgtResetSession
```

Resets the current test configuration to default values

```
AgtSetSessionLabel SessionHandle SessionLabel
Assigns a descriptive label to a test session
AgtGetSessionLabel SessionHandle -> SessionLabel
AgtGetSessionType SessionHandle -> SessionType
AgtGetSessionPid SessionHandle -> SessionPid
Returns a test session's label, type, and process ID
```

To manage tests remotely

```
AgtSetServerHostname ?Hostname?
```

Selects the tester (host name or IP address) to which subsequent commands apply

```
AgtGetServerHostname -> Hostname
```

Returns the hostname or IP address of the currently active tester

To manage test objects

```
AgtListObjects ?AGT_SAVEABLE? ?AGT_SAVED <Filename>? ->
Objects
```

Lists the tester objects you may program through AgtInvoke, may save into a configuration file, or previously saved in a configuration file.

```
AgtInvoke Object Method InParam1 InParam2 -> OutParam
```

Accesses the specified tester object and controls it as indicated by the method (get, set, enable, list, etc.) and parameters (see Quick Reference for all available objects).

```
AgtFormatTime SystemTimeSec -> FormattedTime
```

Returns a formatted time string for a time expressed in seconds since Epoch.

To debug scripts

`AgtBreakPoint`

Interactively steps through each AgtInvoke command in a script, allowing you to isolate the source of problems.

AgtBreakPoint

Syntax

```
AgtBreakPoint
```

Summary

Interactively steps through each AgtInvoke command in a Tcl script, allowing you to view the results of each command and isolate the source of problems.

Details

When you source an API script, the commands are executed one after the other until the script either runs successfully to completion or generates an error. If there is an error, you can use AgtBreakPoint to isolate the command that yielded the error by checking the results of commands leading up to the error.

Embed AgtBreakPoint before the problematic area within the script, such that it prompts you only about the commands in question. (This command is not as useful when entering commands one at a time through the Tcl shell.)

When an AgtBreakPoint command is encountered in a script, the following prompt is displayed in the Tcl shell:

```
About to perform 'AgtInvoke <Object> <Method> '
(<CR>,y,n,q,c,t)
```

The possible actions are:

- <CR> or y: Execute the next command.
- c: Continue running the script to completion, without breaking with prompts.
- n: Don't execute this command.
- q: Exit this script.
- t: Trace this command (not currently implemented).

Error codes

0 Success.
1 Bad argument.

Example Inserting an AgtBreakPoint command into the sample script basic-test.tcl before the first AgtInvoke command yields the prompts shown below. Press <CR> to continue after each prompt.

```
% source basic-test.tcl
```

Setting up the basic test ...

```
Connecting to session named Administrator on localhost ...

About to perform 'AgtInvoke AgtTestSession ResetSession '
(<CR>,y,n,q,c,t)

Reserving test ports 1A and 1B -- this may take several
seconds ...

About to perform 'AgtInvoke AgtPortSelector AddPorts {1A
1B}'(<CR>,y,n,q,c,t)

Setting the SUT interface IP addresses to 192.18.1.1/24 and
192.18.2.1/24 ...
About to perform 'AgtInvoke AgtSutInterfaceList Add
AGT_SUT_INTERFACE' (<CR>,y,n,q,c,t)

About to perform 'AgtInvoke AgtSutInterfaceList Add
AGT_SUT_INTERFACE' (<CR>,y,n,q,c,t)

About to perform 'AgtInvoke AgtSutInterface SetSutIpAddress
1 192.18.1.1 24' (<CR>,y,n,q,c,t)

About to perform 'AgtInvoke AgtSutInterface SetSutIpAddress
2 192.18.2.1 24' (<CR>,y,n,q,c,t)
```

AgtCloseSession

Syntax

```
AgtCloseSession SessionHandle
```

Synopsis

Closes a running test session.

Notes:

- You cannot close a session if a GUI is still connected, but can if only API clients are connected.
- If this command does not work, see “AgtKillSession” on page 61.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtOpenSession or AgtListOpenSessions.

Details

You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection. AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Error codes

```
0 Success.

1 Invalid session handle. Use AgtListOpenSessions to obtain
handles for currently running sessions, and
AgtGetSessionLabel to identify each session.
```

Example

To open a test session through the API

AgtConnect

Syntax

```
AgtConnect ?Session? -> ConnectionID
```

Synopsis

Opens a TCP socket connection between your API client (ie. the Tcl shell interpreting your script or interactive commands) and a currently running test session. Allows you to begin sending commands to a test session.

Note: If you used AgtOpenSession to start a test session, you do not need to call this command. Your API client will be connected automatically. You use this command to connect to test sessions launched through the graphical user interface or by other API clients. Use AgtListOpenSessions to list the currently running test sessions. You may connect to sessions opened by a GUI as a hosted DLL, as well as those opened by a GUI or another API client as a standalone EXE executable.

Parameters

Session	long	(Optional) An ID number or a label to a currently running test session. To list the IDs of current sessions, use AgtListOpenSessions. To get the label of a current session, use AgtGetSessionLabel. You may leave out this parameter if there is only one test session currently running. If however there are no or multiple sessions running, you will get an error. To connect to a test session on a remote test system, simply call AgtSetServerHostname beforehand to specify the remote test system to which this AgtConnect call applies.
ConnectionID	long	A handle to the connection, which becomes the current connection to which subsequent commands apply. You may have multiple connections to several test sessions, but there is always only one active connection. Switch the active connection using AgtSetActiveConnection.

Details

You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using

AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection.

AgtCloseSession closes a currently open test session, and any connection to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Error codes

0 Success.

1 Invalid session handle.

- If you specified a session: The session is not running.
- If you did not specify a session: There is no session running or there are more than one.

Examples

- To connect an API client to a test session
- To open a test session through the API
- To manage tests from a remote computer

AgtDisconnect

Syntax `AgtDisconnect ?ConnectionID?`

Synopsis Closes the specified connection.

Parameters

Connection ID	long	(Optional) A handle to a connection, as returned by AgtConnect or AgtOpenSession. If not specified, defaults to the active connection, as last set by AgtConnect, AgtOpenSession, or AgtSetActiveConnection. To determine the current, active connection, call AgtGetActiveConnection.

Error codes 0 Success.

1 Invalid connection.

- If you specified a connection: There is no such connection active.
- If you did not specify a connection: There is no active connection.

Example To stop a test

AgtFormatTime

Syntax

```
AgtFormatTime SystemTimeSec -> FormattedTime
```

Synopsis

Accepts a time value in terms of the number of seconds elapsed since an epoch, and returns it in a date and time-of-day format.

Parameters

SystemTime	integer	System time in terms of the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time according to the system clock.
FormattedTime	list	System time formatted as: [Abbreviated Day Name] [Abbreviated Month Name] [Day of Month] [Hour:Minute:Second] [Year with Century]. For example, Sun Oct 31 13:46:50 1999

Details

Objects such as AgtIpStatus, AgtHdlcStatus, and AgtSonetStatus can obtain a timestamped record of when events occurred. The recorded system time is in terms of the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (according to the system clock). AgtFormatTime formats this system time into the readable date and time-of-day format.

Error codes

0 Success.

1 There are no active connections.

Example

```
set statusHistory [AgtInvoke AgtSonetStatus
GetStatusHistory]
set systemTime [lindex $statusHistory 3]
set formattedTime [AgtFormatTime $systemTime]
```

AgtGetActiveConnection

Syntax `AgtGetActiveConnection -> ConnectionID`

Synopsis Returns the API client's active connection, to which subsequent commands apply.

Parameters

ConnectionID	long	A connection ID previously returned by AgtConnect or AgtOpenSession.

Details An API client connects to a running test session using AgtConnect, or creates a connection to a new session using AgtOpenSession. A client can maintain multiple connections to different test sessions but there is always only one active connection. The most recent call to AgtConnect or AgtOpenSession sets the active connection implicitly. Use AgtGetActiveConnection to determine the active connection and AgtSetActiveConnection to select the active connection explicitly. When finished, use AgtDisconnect to close a connection.

Error codes 0 Success.

1 There are no active connections.

Example To open a test session through the API

AgtGetServerHostname

Syntax `AgtGetServerHostname -> Hostname`

Synopsis Returns the host name of the test system to which the Tcl client is currently connected.

Parameters

Hostname	string	The host computer name of the test system to which subsequent commands apply. You can determine a test system's host computer name and IP address by right-clicking Network Neighborhood and selecting Properties.

Details The test system software itself always runs on the Windows computer that is connected to the test modules. However, you may control the system remotely, from another Windows computer or UNIX-based workstation connected to the test system via TCP/IP.

You can control the test system as part of a larger test configuration comprising other test instruments and scripts which control the system under test. You simply copy the small library of portable Tcl commands, AgtClient, to the remote computer. By default, the scripting environment selects "localhost" and Tcl scripts and commands operate on the local computer. To operate on a remote computer, call AgtSetServerHostname. To determine the computer to which a script is currently connected, call AgtGetServerHostname.

Error codes 0 Success.

Example To manage tests from a remote computer

AgtGetSessionLabel

Syntax `AgtGetSessionLabel SessionHandle -> SessionLabel`

Synopsis Returns the descriptive label assigned to the specified (running) test session, to help identify that session amongst multiple running sessions.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtListOpenSessions or AgtOpenSession.
SessionLabel	string	A description of the test session. The default label is the user login name for sessions launched through the GUI and "SYSTEM" for sessions launched through the API. This name may be changed using AgtSetSessionLabel.

Details The test session must be running, either from the graphical user interface or from a call to AgtOpenSession.

Error codes 0 Success.

1 Invalid session handle. Use AgtListOpenSessions to obtain valid handles for currently running sessions, and AgtGetSessionLabel to get the label currently used to identify each session.

Example To open a test session through the API.

AgtGetSessionPid

Syntax

```
AgtGetSessionPid SessionHandle -> SessionPid
```

Summary

Returns a test session's process ID.

Note: This is the same as AgtInvoke AgtSessionManager GetSessionPid.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtOpenSession or AgtListOpenSessions.
SessionPid	long	If there are several users running test sessions and you need to kill a session through the Windows Task Manager, you can use this to identify a session's process ID.

Details

For information about test sessions, see AgtOpenSession (Details).

Error codes

```
0 Success.
1 Invalid session handle. Use AgtListOpenSessions to obtain
handles for currently running sessions, and
AgtGetSessionLabel to identify each session.
```

Example

```
% AgtListOpenSessions
5
% AgtGetSessionPid 5
142
```

AgtGetSessionType

Syntax

```
AgtGetSessionType SessionHandle -> SessionType
```

Summary

Returns a test sessions's type.

Note: This is the same as AgtInvoke AgtSessionManager GetSessionType.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtOpenSession or AgtListOpenSessions.
SessionType	string	The type of test session. For a listing of possible types, see AgtListSessionTypes.

Details

For information about test sessions, see AgtOpenSession (Details).

Error codes

```
0 Success.
1 Invalid session handle. Use AgtListOpenSessions to obtain
handles for currently running sessions, and
AgtGetSessionLabel to identify each session.
```

Example

```
% AgtListOpenSessions
5
% AgtGetSessionType 5
IpPerformance
```

AgtGetVersion

Syntax

```
AgtGetVersion -> Version
```

Synopsis

Gets the version of the XM software currently installed on a local or remote PC.

Parameters

Version	string	The version of the XM base software currently installed on the PC. Includes a major and minor version number (eg. 1.2). May also include two additional numbers if you are using pre-release software (eg. 1.1.4.11).

Details

To get the version of the XM software on a remote PC, you must first identify the remote PC — see the example below.

Error codes

0	Success.
1	Unable to connect to the session manager. The PC either does not have the XM software installed or does not have its Resource Manager service running

Example

% AgtGetVersion
1.2
% AgtSetServerHostname OmniBERXM_2
OmniBERXM_2
% AgtGetVersion
1.1.4.11

.

AgtInvoke

Syntax `AgtInvoke Object Method ParameterList`

Synopsis Sends messages to objects in the test system (the same objects represented in the graphical user interface), so that you can configure, start/stop, and get information about test system components.

Parameters

Object	<object>	An object in the test system, as listed under the Quick Reference in Objects.
Method	<method>	One of several possible actions you can perform on the object.
ParameterLis t	list <params>	The parameters required by or returned by the method.

Details Once a connection is established with a running test session (using AgtConnect or AgtOpenSession), AgtInvoke is the primary API command used to interact with the tester. The functionality of the tester is provided through objects. Each object provides a number of methods, which are valid operations on the object. Each method in turn has parameters: some input, some output.

Error codes
0 Success.
1 Bad argument.

Examples To connect an API client to a test session

AgtKillSession

Syntax

```
AgtKillSession SessionHandle
```

Synopsis

Closes a running test session if AgtCloseSession cannot — please read the Details below.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtListOpenSessions or AgtOpenSession.

Details

Normally, you use AgtCloseSession to close a test session. But, there may be a specific situation where you cannot connect to a running test session to close it or AgtCloseSession cannot close the session.

If the test session is running on a remote computer, use AgtSetServerHostname to connect to that computer first, before issuing the command.

This command uses the "kill.exe" program to end the session's process. The program bypasses the Windows Task Manager requirement that you have administrators' privileges to end OMniBER XM processes.

Error codes

0	Success
1	Invalid session handle. Use AgtListOpenSessions to obtain valid handles for currently running sessions, and AgtGetSessionLabel to get the label currently used to identify each session.

Example

```
% package require AgtClient
0.1
% AgtSetServerHostname omniberxm
omniberxm
% AgtListOpenSessions
6
% AgtKillSession 6
```

AgtListConnections

Syntax `AgtListConnections -> ConnectionIDs`

Synopsis Lists all the connections that are open between the current API client (ie. the Tcl shell interpreting your script or interactive commands) and local or remote test sessions.

Parameters

ConnectionIDs	list<long>	A list of connection IDs. These IDs were previously returned by AgtConnect or AgtOpenSession.

Details To determine which of the listed connections is the active one (ie. the one to which subsequent commands apply), call AgtGetActiveConnection. To set the active connection to another one, call AgtSetActiveConnection.

Error codes 0 Success.

Example To open a test session through the API

AgtListObjects

Syntax

```
AgtListObjects ?AGT_SAVEABLE? ?AGT_SAVED <Filename>? ->
Objects
```

Synopsis

Lists the API objects that you may program (varies depending on the test modules and applications purchased for your tester). Or, lists the objects whose configuration settings may be saved through AgtSaveSession.

Parameters

AGT_SAVEABLE	enum	(Optional) Include this enumerated value to list only those API objects whose configuration settings are saveable through AgtSaveSession.
AGT_SAVED	enum	(Optional) Include this enumerated value to list the API objects saved in the specified configuration file.
Filename	string	(Required if using AGT_SAVED) The name of the configuration file whose saved objects you want to list. The file name should be of a fully specified pathname, as described for AgtSaveSession.
Objects	string	A list of the API objects available or saveable.

Details

The tester components you may program are represented by objects, for example, types of protocol emulations, simulated network topologies, traffic characteristics, statistics. Each object has its own set of methods, representing the actions you may perform, for example, set or get. You use this command to list the objects that are available on your tester. For a complete list of all available objects, see this quick reference to the API objects. Use the command AgtInvoke to invoke methods on objects.

Error codes

0 Success.

Examples

- To connect an API client to a test session
- To save, restore, or reset a test configuration

AgtListOpenSessions

Syntax `AgtListOpenSessions -> SessionHandles`

Synopsis Lists the test sessions that are currently open.

Parameters

SessionHandles	list<long>	A list of handles to the open test sessions.

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection. AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Error codes `0 Success.`

Example To open a test session through the API

AgtListSessionTypes

Syntax `AgtListSessionTypes -> SessionTypes`

Summary Lists all test applications available on the OmniBER XM system.

NOTE You must enter a valid license to use optional applications. For details, see the User's Guide, Chapter 1, "Introduction", "To add or change product licenses".

Parameters

SessionTypes	string	The type of test session. For a listing of possible types, see "AgtOpenSession" on page 66.

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession.

You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection.

AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Error codes 0 Success.

Example To open a test session through the API

AgtOpenSession

Syntax `AgtOpenSession SessionType ?SessionMode? -> SessionHandle`

Summary Starts an instantiation of the test session software.

Parameters

SessionType	string	A type of test application. Yiu may pass AgtOpenSession to open a test session.
SessionMode	string	(Optional) Whether to start a session for full testing or configuration only. • AGT_SESSION_ONLINE: (Default, selected if this parameter is not specified) You select this mode if you are doing more than just configuring a test, for example, if you also want to generate traffic and view statistics. This reserves the modules you select on a subsequent call to AgtPortSelector, thus locking out anyone else who might attempt to use the same modules to generate traffic. The test session will actively connect to all selected test ports and download test configurations. Note: The test modules do not need to be connected as you can simulate connected test modules using the system variable AGT_DUMMY_MODULES. • AGT_SESSION_OFFLINE: You select this mode if you are simply configuring tests. This neither reserves the test modules nor locks out anyone who may want to use the GUI or API to run tests on the same modules. Your test configurations are stored locally on the PC and not downloaded to the test ports. Note: API clients cannot set the test session's context (that is, EXE or DLL), the way you can through the GUI. All test sessions launched through the API are launched as detached EXE executables. To get the mode and context of the current test session, use AgtInvoke AgtTestSession (GetMode and GetContext methods).
SessionHandle (ConnectionID)	long	A handle to the test session, which doubles as the handle to the connection that is automatically set up between the API client and the test session. This connection becomes the current connection to which subsequent commands apply. To switch to another connection, use AgtSetActiveConnection. To get the handle of the current test session, use AgtInvoke AgtTestSession (GetHandle method).

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession.

You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection.

AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Error codes

```
0    Success.
1    Bad argument.
```

Example To open a test session through the API

AgtResetSession

Syntax `AgtResetSession ?Objects?`

Synopsis Resets the current test session configuration, restoring all saveable objects to their default state.

Parameters

Objects	string OR list<string>	(Optional) A list of the objects whose settings you want to reset. By default, if this parameter is left out, all objects are reset. To list the objects you can reset, use the command AgtListObjects with the parameter AGT_SAVEABLE. To build a list of the objects you want to reset, see the Example.

Details This is useful at the end of a test. You do not have to explicitly reset the configuration to original values.

Error codes

0	Success.
1	Error.

Example To save, restore, or reset a test configuration

AgtRestoreSession

Syntax AgtRestoreSession Filename ?Objects?

Synopsis Restores a test session to a previously saved configuration. You may restore all objects saved in the file or *specific objects* only.

Parameters

Filename	string	The name of the file from which to restore the configuration. If a test session is currently active, the default directory is "c:/Program Files/Agilent/OmniBER XM/config/<SessionType>". If you want to restore a file from another location or if a test session is not currently active, you must specify the full directory path including the drive designator (c:/). For details, see AgtSaveSession. If a test session was not active, one is opened using the specified configuration.
Objects	string OR list<string>	(Optional) A list of the objects whose settings you want to restore. By default, if this parameter is left out, all objects are restored. To list the objects saved in a file, use the command AgtListObjects. To build a list of the objects you want to save, see the Example.

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using AgtConnect and disconnect using AgtDisconnect.

AgtSaveSession can be used to save the configuration of the test session to file. The configuration can then be restored from the given file, at a later time, using the AgtRestoreSession command. If there is no running test session, AgtRestoreSession opens a new test session and restores its configuration from the supplied file. If there is a connection to a running session, then AgtRestoreSession restores the configuration of that session.

Error codes

0 Success.
1 Bad argument.

Example To save, restore, or reset a test configuration

AgtSaveSession

Syntax `AgtSaveSession Filename ?Objects?`

Synopsis Saves the current test configuration to a plain text file. Lets you quickly restore the tester to the same configuration, make configuration changes through the text file (eg. quick setup of many SUT IP addresses by cutting and pasting from another source), or debug problems by looking for anomalies in the text file. You may save all test settings or those for specific objects only (eg. to restore or debug only a part of a test).

Parameters

Filename	string	The name of the file to which to save the configuration. Use the file extension .xml xml. By default, the file is saved in the directory "c:/program files/agilent/omniberxm/config/<SessionType>/". To save in another location, specify the full directory path along with the drive designator (c:/). Examples of valid file names. • config.xml • c:/temp/config.xml • c:/progra~1/agilent/omniberxm~1/config/ipperf~1/config.xml • "c:/program files/agilent/omniber xm/config/ipperformance/config.xml" Rules when specifying directory paths: • create the directory first if it doesn't already exist (non-existent ones will not be created automatically) • the drive and directory names are not case sensitive • if a directory name contains a space or has > 8 characters, use double quotes to enclose the whole directory path and file name ("x") • shortcut: for directory names with spaces, remove the space and append "~1" to the end of the name (eg. for "tst cf", specify "tstcf~1")

		• shortcut: for directory names with > 8 characters, truncate to 6 characters (after removing the spaces) and append "~1" (eg. for "my configs", specify "myconf~1"); the ~1 (then ~2, ~3, etc.) is used to uniquely identify directories with the same root name. • if using double quotes to enclose a directory path, you may use forward slashes (/) or backslashes (\) to separate directories; if not using double quotes, you must use forward slashes (/) Rules when specifying file names: • new file names are case sensitive (existing ones are not) • files that already exist will be overwritten without warning • names may be more than 8.3 characters long • if the file name contains a space, use double quotes to enclose the whole directory path and file name ("x")
Objects	string OR list<string>	(Optional) A list of the objects whose settings you want to save. By default, if this parameter is left out, all saveable objects are saved. To list the saveable objects, use the command AgtListObjects. To build a list of the objects you want to save, see the Example.

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using AgtConnect and disconnect using AgtDisconnect.

AgtSaveSession can be used to save the configuration of the test session to text file, to ease the setup of future tests. The state of the test session can then be restored from the given file at a later time using the AgtRestoreSession command.

Error codes 0 Success.

1 Possible errors:

- Unable to open file for writing. Make sure the directory path and file name adhere to the rules listed for the Filename parameter.
- Unable to save session while test is running. You must stop a test before you can save its configuration.

Example To save, restore, or reset a test configuration

AgtSetActiveConnection

Syntax `AgtSetActiveConnection ConnectionID`

Synopsis Selects the connection to which subsequent commands apply.

Parameters

ConnectionID	long	A connection ID previously returned by AgtConnect or AgtOpenSession.

Details An API client connects to a running test session using AgtConnect, or creates a connection to a new session using AgtOpenSession. A client can maintain multiple connections to different test sessions but there is always only one active connection. The most recent call to AgtConnect or AgtOpenSession sets the active connection implicitly. Use AgtGetActiveConnection to determine the active connection and AgtSetActiveConnection to select the active connection explicitly. When finished, use AgtDisconnect to close a connection.

Error codes 0 Success.

1 Invalid Connection ID. Connection ID must be as a result of a AgtConnect or AgtOpenSession command.

Example To open a test session through the API

AgtSetServerHostname

Syntax `AgtSetServerHostname ?Hostname?`

Synopsis Selects the local or remote test system to which subsequent commands apply.

Parameters

Hostname	string	(Optional) The host computer name of the test system to which subsequent commands apply. You can determine a test system's host computer name by right-clicking Network Neighborhood and selecting Properties. You may also specify the host's IP address. To determine a test system's IP address, right-click Network Neighborhood, select Properties, select the Protocols tab, and click Properties. By default, if you leave out this parameter, the scripting environment selects "localhost" and Tcl scripts operate on the local test system.

Details The test system software itself always runs on the Windows computer that is connected to the test modules. However, you may control the system remotely, from another Windows computer or UNIX-based workstation connected to the test system via TCP/IP.

You can control the test system as part of a larger test configuration comprising other test instruments and scripts which control the system under test. You simply copy the small library of portable Tcl commands, AgtClient, to the remote computer. By default, the scripting environment selects "localhost" and Tcl scripts and commands operate on the local

computer. To operate on a remote computer, call AgtSetServerHostname. To determine the computer to which a script is currently connected, call AgtGetServerHostname.

Error codes

0 Success.
1 Unable to change hostname. Invalid host.

Example To open a test session through the API

AgtSetSessionLabel

Syntax

```
AgtSetSessionLabel SessionHandle SessionLabel
```

Synopsis

Assigns a descriptive label to a test session, to identify that session amongst several running sessions.

Parameters

SessionHandle	long	A handle to the test session, as returned by AgtListOpenSessions or AgtOpenSession.
SessionLabel	string	A descriptive label for the test session. The default label is the name of the user who opened the session; check the current label using AgtGetSessionLabel. Use double quotes (") to enclose strings containing spaces.

Details

If the test system is simultaneously being used by multiple users, the session label can be used to help identify the owner and/or purpose of the test session.

Error codes

0	Success
1	Invalid session handle. Use AgtListOpenSessions to obtain valid handles for currently running sessions, and AgtGetSessionLabel to get the label currently used to identify each session.

Example

To open a test session through the API

6
Objects

Type Definitions

Type	Description	Example
Unsigned char	An 8-bit numeric value	1
Long	32 bit value, unsigned	12345678
Float	32 bit floating point	1.2345678
Double	64 bit floating point	1.23456e+78
BOOL	Boolean value (TRUE/FALSE)	TRUE
String	String Type	“one two three four five six”
Enumerated Types	Enumerations as a string	AGT_EXAMPLE_ONE
Lists	A list of any of the above types. Defined as a space separated list of items, enclosed by braces - {}. All items within a list must be of the same type	{1 2 3 4} {1.0 2.0 3.0 4.0} {TRUE FALSE TRUE FALSE} {“one” “two” “three” “four”} {ONE TWO THREE FOUR}

Quick Reference

Manage sessions

Manages the creation, connection/disconnection of test sessions.

Select ports

Reserves test ports for the current test session. As ports are added, modules are locked for exclusive use by the session.

Control tests

Sets general parameters for a test, starts and stops testing, and gets test information.

Control lasers

Turns on/off optical transmit lasers, selects a transmit/receive mode, and selects the clock source.

Log statistics

Records selected statistics to a text file in real time.

AgtTestController

Syntax

```
AgtInvoke AgtTestController Method InParams -> OutParams
```

Methods

```
SetTestMode Mode
GetTestMode -> Mode

SetTestDuration Duration
GetTestDuration -> Duration

SetSamplingInterval SamplingInterval
GetSamplingInterval -> SamplingInterval
GetSamplingIntervalLimits -> MinSeconds MaxSeconds

SetTrickleTime TrickleTime
GetTrickleTime -> TrickleTime
GetTrickleTimeLimits -> MinSeconds MaxSeconds

StartTest
StopTest

GetTestState -> TestState
GetStartTime -> StartTime
GetElapsedTime -> ElapsedTime
```

Synopsis

Sets general parameters for a test; starts and stops testing; and gets test information.

Parameters

Mode	Enum	Whether to test continuously or for a fixed duration: • AGT_TEST_CONTINUOUS: (Default) Run the test until stopped, through the StopTest method or the Stop button on the graphical user interface. • AGT_TEST_ONCE: Run the test for the given Duration, or until the test is stopped (whichever occurs first). You cannot change this mode during a test; the test state must be AGT_TEST_STOPPED.
Duration	long	Applies only for the AGT_TEST_ONCE test mode) The duration of the test, in seconds. May range from 1 to 604,800 seconds (that is, a week); default is 60 seconds. You cannot change this value during a test; the test state must be AGT_TEST_STOPPED.

MinSeconds MaxSeconds	long	The minimum and maximum number of seconds allowed for • GetTestDurationLimits: a test • GetSamplingIntervalLimits: a sampling interval
TestState	Enum	Whether the test is in progress: • AGT_TEST_STOPPED: The test is idle. • AGT_TEST_STARTING: All ports are preparing to start traffic generation and statistics collection synchronously. • AGT_TEST_RUNNING: Traffic generation and statistics collection is in progress. • AGT_TEST_STOPPING: Traffic generation has stopped, but residual statistics collection is continuing for the duration of the trickle time.
StartTime	long	The time of day when the test was started. Allows other timestamps in the system (eg. BufferOverflow, timestamps in capture records, statistics results, etc.) to be correlated with time of day information. The value of StartTime represents the number of seconds since midnight, January 1, 1970. Use AgtFormatTime to convert to date and time-of-day format.
ElapsedTime	long	The time elapsed since the test was started, in seconds.

Details When you start a test, the system synchronously starts generating traffic and measuring statistics across all test ports. The test state progresses to AGT_TEST_STARTING while all ports are being synchronized, and then to AGT_TEST_RUNNING. When you stop a test, the system synchronously stops generating traffic across all test ports. It then stops measuring statistics after the trickle time has elapsed, so that frames in transit can reach their destination test ports and be counted in statistics.

The test state progresses to AGT_TEST_STOPPING when traffic generation stops, and then to AGT_TEST_STOPPED when measurements stop.

There is a timeout of 15 seconds for both StartTest and StopTest. On timeout, a list of the test modules that did not start or stop properly is logged in the event log and the application exits with a fatal error. To disable the timeouts, you must define the environment variable AGT_SUSPEND_TEST_TIMEOUT (assign it any value).

Error Codes

0	Success.
1	Invalid operation: • Test in progress: Cannot change this parameter while there is a test in progress. Stop the test first, using the method StopTest. • Parameter out of range: The value you are trying to set is outside the valid range. • Not implemented: The SetTrickleTime method is not currently implemented.

AgtModuleManager

Syntax

```
AgtInvoke AgtModuleManager Method InParams -> OutParams
```

Methods

```
GetSystemState -> SystemState

RebootAllModules
UpdateModules
DisableAutoUpdate
EnableAutoUpdate
IsAutoUpdateEnabled -> IsEnabled

UseSingleModule SerialNumber
ListModules -> ModuleNumbers
ListAllModules -> SerialNumbers
GetSerialNumber ModuleNumber -> SerialNumber
GetModuleNumber SerialNumber -> ModuleNumber
GetModuleName SerialNumber -> ModuleName
GetNamedModule ModuleName -> SerialNumber
SaveModuleList
GetSavedModuleList -> SerialNumbers
ListNewModules -> SerialNumbers
ListMissingModules -> SerialNumbers

GetModuleDescription SerialNumber -> ModuleType
PortsInModule
GetPortType SerialNumber PortNumber -> PortType
GetPortName SerialNumber PortNumber -> PortName
GetNamedPort SerialNumber PortName -> PortNumber
GetNcpCount SerialNumber -> NcpCount
IsModuleSynchronized SerialNumber -> IsSynchronized
IsDummyModule SerialNumber -> IsDummyModule

IsChassisBlade SerialNumber -> IsChassisBlade
GetChassisNumber SerialNumber -> ChassisNumber
GetChassisSlotNumber SerialNumber -> ChassisSlotNumber
SetModuleAnnotation SerialNumber ModuleAnnotation
GetModuleAnnotation SerialNumber -> ModuleAnnotation
IsModuleClockMaster SerialNumber -> ModuleClockMaster

GetIpAddress SerialNumber NcpIndex -> IpAddress
GetPrimaryIpAddress SerialNumber -> IpAddress
GetHostIpAddress -> IpAddress
GetModuleState SerialNumber -> ModuleState
GetModuleLock SerialNumber -> SessionHandle
UnlockModule SerialNumber
RebootModule SerialNumber


IsShutdownRequired SerialNumber -> ShutdownRequired
ShutdownModule SerialNumber
```

```
FlashModuleLEDs SerialNumber
ShowIpAddresses
```

Summary Manages test modules, allowing you to perform the same diagnostics and troubleshooting possible through the OmniBER XM Diagnostics Tool.

Note: It is easier to diagnose problems through the Diagnostics Tool, since it provides a visual, at-a-glance summary of the status of the tester and its modules. This API support allows remote, automated, customized diagnostics and provides a few additional functions (indicated above by *). See Details for more information about the supported methods.

Parameters

SystemState	enum	The current state of the tester: • AGT_SYSTEM_READY: The tester does not detect any problems with its connected test modules. • AGT_SYSTEM_UPDATE_PENDING: The tester is waiting to update the module numbers. When the tester detects a new module, it waits for 10 seconds before numbering the modules. Also, the tester cannot renumber the modules if a test session has locked (that is, is using) any modules. In this case, the tester waits until all test sessions are closed before renumbering the module numbers. For details, see To troubleshoot module problems, A module's MODULE LED is blank. • AGT_SYSTEM_UPDATING: The tester is in the process of numbering its connected test modules. • AGT_SYSTEM_UPDATE_FAILED: At least one module could not be assigned an ID number. See also To troubleshoot module problems.

IsEnabled	bool	Indicates whether automatic module numbering is enabled: 1 (default): Automatic numbering is on. This means the Module Manager periodically checks for newly added or removed test modules and renumbers all the test modules accordingly. This must be enabled for plug-and-play module operation. 0: Automatic numbering is off. This is required to use the method UseSingleModule, which uses one specific module only and assigns that module the number 1.
SerialNumber	string	The serial number of a test module. This number is shown on a sticker at the rear of the module. If you are simulating modules through Demo mode, the number is "AGT_MODULE_x" where x describes the type of module being simulated. You can also obtain serial numbers through AgtModuleManager, by calling the methods ListAllModules or GetSerialNumber.
ModuleNumbers	list<long>	A list of numbers identifying the test modules that are currently connected and powered up. Modules are assigned incremental numbers, according to their order on the Event daisy chain.
SerialNumbers	list<string>	A list of module serial numbers. A module's serial number is shown on the physical test module, on a sticker on the back panel. If you are simulating modules through Demo mode, the number is "AGT_MODULE_x" where x describes the type of module being simulated.
ModuleNumber	long string	See AgtPortSelector for details about these Parameters.
ModuleName	enum	
ModuleType	long	
PortNumber	string	
PortName	enum	
PortType		
PortsInModule	long	The number of test ports in the test module. Use this value if you do not want to determine the value by checking against the above list of possible module types.

IsSynchronized	bool	Indicates whether a test module needs to be synchronized with its neighbors: 1: yes 0: no
IsChassisBlade	bool	Set to TRUE if the module is an OmniBER module in a chassis.
ChassisNumber	long	The chassis number you wish to lock the event line to.
ChassisSlotNumber	long	Returns the slot number the card is in. Note you can also work this out from the module number itself. For example, card 101 is in slot 1.
ModuleAnnotation	string	A string describing a module in the registry.
IsDummyModule	bool	Indicates whether a test module is being simulated through Demo mode. 1: This module is being simulated. 0: This module is not being simulated.
IpAddress	string	The IP address of the first test port (Port A) in the test module. The PC uses a DHCP server to assign IP addresses dynamically to its test ports. By default, the tester Ethernet card that connects the test modules uses the IP address of 10.0.0.1 (subnet mask 255.0.0.0) and assigns addresses within this subnet, starting with 10.0.0.2. You may use different IP addresses if these addresses interfere with addresses being used in your test lab or corporate LAN. For details, see the User Guide, “To Change the IP Address of the Hub Card”.

ModuleState	enum	The current state of the test module: • AGT_MODULE_READY: The module booted successfully. Check the system state for an Update pending condition. • AGT_MODULE_LOCKED: The module is in use and locked out by a test session. **Note:** If modules are currently locked by a test session, newly connected test modules will not be able to determine their module ID numbers. The Clock and Event lines are required to do this. The new modules will have Module IDs of zero. To renumber the modules, close the test session that is locking the other modules. • AGT_MODULE_REBOOTING: The module is in the process of rebooting and should be available soon. A module automatically reboots after you unselect its ports from a test session or close its test session. It might take up to a minute for a module to reboot. If it remains in the Rebooting state for more than this, it may be failing its boot process. Check the module's physical MODULE LED.
SessionHandle	long	A handle to the test session that is currently using (that is, locking out) a test module.
ShutdownRequired	bool	Indicates whether a device's operating system needs to be shut down before it can be powered off 1: yes 0: no :OmniBER XM modules do not need to be shut down.

Details AgtModuleManager allows you to:

- get the current state of the tester
- reboot all modules or a single module (Note: you cannot reboot a module if it is currently in use by a test session)
- update module numbers (for example, after adding or removing modules)

- disable automatic module numbering (for example, to use a single module and assign it the number 1)
- use a single module for test purposes (Note: you must disable automatic numbering, the module cannot be currently in use by a test session)
- list the module or serial numbers of modules
- get a specific module's module or serial number
- save the current list of modules found in the system to the registry
- return the saved list of modules
- list the new modules that were not present the last time the list of modules was saved
- list the missing modules that were present the last time the list of modules was saved
- determine whether a module has the master clock, is a dummy (simulated) module
- get the IP addresses of a modules' test ports
- get the current state of a module
- determine which test session has a lock on a module
- unlock a module that was not unlocked properly after its session closed
- locate a module in a physical stack of modules, by either flashing its module LED or scrolling the IP address of its test port A across the LED

Details about individual methods:

GetSystemState: Return the current state of the system: AGT_SYSTEM_READY, AGT_SYSTEM_UPDATE_PENDING, AGT_SYSTEM_UPDATING. Modules can only be locked when the system is in the READY state.

RebootAllModules: Reboot all modules. All modules must be unlocked first.

UpdateModules: This command is issued to instruct the ModuleManager to reallocate module numbers as a result of modules being added and removed. If any modules are locked or rebooting, the update will be deferred until the modules are either ready or marked as failed.

DisableAutoUpdate: Disable automatic updating of module numbers. The Module Manager will not automatically assign module numbers.

EnableAutoUpdate: Enable automatic updating of module numbers.

IsAutoUpdateEnabled: Is automatic updating of module numbers enabled?

UseSingleModule: Instead of assigning module numbers to all modules, assign the number 1 to a single module. If automatic updating is enabled, or any modules are locked, this request will fail. For manufacturing test.

ListModules: List all numbered modules.

ListAllModules: Return a list of all modules found in the test system.

GetSerialNumber: Return the serial number of a numbered module.

GetModuleNumber: Return the module number assigned to a module. If no module number is assigned, return zero.

GetModuleName: Return the name of a module. May be blank.

GetNamedModule: Return the serial number of the module with the specified name, provided that it is unique.

SaveModuleList: Saves the current list of modules found in the system to the registry.

GetSavedModuleList: Return the saved list of modules.

ListNewModules: List modules in the system which were not there the last time the list of modules was saved.

ListMissingModules: List modules which were in the system the last time the module was saved, which are not there now.

GetModuleDescription: Returns the type of the module and the number of ports available in that module. Providing the port count enables scripting clients to discover the number of ports available without having to hardcode details of specific module types.

GetPortType: Returns the physical interface type for the port (indexed from 1 to N).

GetPortName: Returns the name of the port (may be blank).

GetNamedPort: Return the number of the port with the specified name.

GetNcpCount: Returns the number of NCPs in the module. Each NCP may control more than one port.

IsModuleSynchronized: Return TRUE if the module is synchronized with other modules.

IsDummyModule: Returns false if module is not a dummy module.

GetIpAddress: Returns the IP address of the given NCP.

GetPrimaryIpAddress: Returns the IP address of the primary NCP.

GetHostIpAddress: Returns the IP address of the host on the OmniBER XM network.

GetModuleState: Return the current state of the given module.

GetModuleLock: Returns the session handle currently lock the module for a given serial number.

UnlockModule: Force unlock of a module Used to provide a mechanism so if for some reason the locks held by a particular test session are not cleaned up.

RebootModule: Reboot a module. The module must be unlocked. Used to provide a mechanism to reboot a module for diagnostic purposes if it doesn't reboot automatically.

IsShutdownRequired: Check whether a module needs to be shut down before powering off.

ShutdownModule: Shut down a module before powering it off. The module will enter the AGT_MODULE_SHUTTING_DOWN state; once it enters the AGT_MODULE_SHUTDOWN state, it's safe to power off.

FlashModuleLEDs: Flashes power LEDs to yellow at 2 Hz for 5 seconds for module with given serial number.

ShowIpAddresses: Show the IP addresses of all modules on their LED displays.

Error codes

0 Success.

> 0 Bad argument.

Example

```
% AgtInvoke AgtModuleManager GetSystemState
AGT_SYSTEM_READY
% AgtInvoke AgtModuleManager ListModules
1 2 3 4
% AgtInvoke AgtModuleManager ListAllModules
AU12345678 AU23456789
```

AgtSessionManager

Syntax

```
AgtInvoke AgtSessionManager Method InParams -> OutParams
```

Methods

```
OpenSession SessionType SessionMode -> SessionHandle
CloseSession SessionHandle

ListSessionTypes -> SessionTypes
ListOpenSessions -> SessionHandles

GetSessionType SessionHandle -> SessionType
GetSessionPort SessionHandle -> SessionPort
GetSessionContext SessionHandle -> Context

SetSessionLabel SessionHandle SessionLabel
GetSessionLabel SessionHandle -> SessionLabel

GetSessionPid SessionHandle -> ProcessId

GetNumGuiConnections SessionHandle SessionType ->
NumConnected
GetMaxGuiConnections SessionType -> MaxConnections
SetMaxGuiConnections SessionType MaxConnections
```

Summary

Manages test sessions. **Note:** Most of these methods operate on the **current** test session. To change the current session, call AgtSetActiveConnection and pass the desired session handle. (The session's handle number is also used as the connection's ID number.) To list the handles for the active sessions, call AgtListOpenSessions.

Parameters

SessionType	string	The type of test session. For a listing of possible types, see AgtOpenSession.
SessionMode	enum	Whether a session is being used for full testing or configuration only. • AGT_SESSION_ONLINE: You select this mode if you are doing more than just configuring a test, for example, if you also want to generate traffic and view statistics. This reserves the modules you select on a subsequent call to AgtPortSelector, thus locking out anyone else who might attempt to use the same modules to generate traffic. The test session will actively connect to all selected test ports and download test configurations. Note: The test modules do not need to be connected as you can simulate connected test modules using the system variable AGT_DUMMY_MODULES. • AGT_SESSION_OFFLINE: Select this mode if you are simply configuring tests. This neither reserves the test modules nor locks out anyone who may want to use the GUI or API to run tests on the same modules. Your test configurations are stored locally on the PC and not downloaded to the test ports. The mode you select affects the software launched, is selected when you first open a session, and cannot be changed afterwards
SessionHandle	long	If you called OpenSession, this is a handle to the newly opened session. If you called GetHandle, this is the handle to the **current** test session. To change the current session, call AgtSetActiveConnection and pass the desired session handle. (The session's handle number is also used as the connection's ID number.) To list the handles for the active sessions, call AgtListOpenSessions.

Filename	string	The name of the file used to store the test configuration data. Test configuration files should have the extension **.xml**. The many different rules for specifying the directory path and file name are detailed for AgtSaveSession.
SessionTypes	list<string>	A list of the types of test sessions supported by your tester.
SessionHandles	list<long>	A list of handles to the test sessions currently open.
SessionPort	long	The port number that the session is currently waiting on for scripting connections.
Context	enum	How the current test session was opened. Currently, the only context supported is: • AGT_SESSION_EXE: The session is running as standalone, detached executable program. Advantages: Multiple GUI clients can access the same test session (that is, its test ports, traffic definitions, real-time statistics). You can exit the GUI without closing its test session or terminating any attached GUI or API clients. This context is no longer supported: • AGT_SESSION_DLL: The session is running as a DLL hosted by the GUI. When the GUI closes, the test session closes automatically. Advantage: Test sessions locking test modules are not inadvertently left running in the background.
SessionLabel	string	A descriptive label for the current test session. The default label is • SYSTEM if the session was opened through the API • user login name (for example, "administrator") if opened through the GUI Use double quotes (") to enclose strings containing spaces.

ProcessId	long	If there are several users running test sessions and you need to kill a session through the Windows Task Manager, you can use this to identify a session's process ID.

Details

You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession. You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection

AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

By default, the tester allows one GUI to connect to each test session. This prevents others from accessing and modifying your tests. You can reset this using the SetMaxGuiConnections method.

Error codes

0	Success
1	Bad argument

Example

```
# check the current maximum number of GUI users and set it to
2
% AgtInvoke AgtSessionManager GetMaxGuiConnections
IpPerformance
1
% AgtInvoke AgtSessionManager SetMaxGuiConnections
IpPerformance 2
% AgtInvoke AgtSessionManager GetMaxGuiConnections
IpPerformance
2
% AgtInvoke AgtSessionManager GetNumGuiConnections 1
IpPerformance
1
```

AgtPortSelector

Syntax

```
AgtInvoke AgtPortSelector Method InParams -> OutParams
```

Methods

```
ListModules -> ModuleNumbers
GetLastModule -> Module
GetModuleDescription ModuleNumber -> ModuleType
PortsInModule
GetModuleName ModuleNumber -> ModuleName
IsModuleSynchronized ModuleNumber -> Synchronized
IsChassisBlade ModuleNumber -> IsChassisBlade
GetChassisNumber ModuleNumber -> ChassisNumber
GetChassisSlotNumber ModuleNumber -> ChassisSlotNumber

GetPortType ModuleNumber PortNumber -> PortType
GetPortLabel ModuleNumber PortNumber -> PortLabel
GetPortName ModuleNumber PortNumber -> PortName

ListModuleTypes ModuleNumber -> ModuleTypes
SetModuleType ModuleNumber ModuleType
GetModuleLimit -> Limit
IsModuleSupported ModuleNumber -> IsSupported
ListSessionModuleTypes ModuleNumber -> ModuleTypes

GetModuleState ModuleNumber -> ModuleState
GetModuleLock ModuleNumber -> SessionLock
GetChassisUpstreamLock ChassisNumber -> SessionLock
GetChassisDownstreamLock ChassisNumber -> SessionLock
GetLockedModuleList -> ModuleNumbers
GetLockedModules -> FirstModule LastModule
ListRequiredModules SelectedModules -> RequiredModules
ListUnavailableModules SelectedModules -> UnavailableModules

AddPort ModuleNumber PortNumber -> Handle
AddPorts PortLabels -> PortHandles
AddNamedPort ModuleName PortName -> PortHandle
AddPortsWithLock PortLabels ModuleNumbers -> PortHandles
RemovePort PortHandle
RemovePorts PortHandles
ListPorts -> PortHandles
FindPortHandle ModuleNumber PortNumber -> PortHandle
FindPortHandleFromLabel PortLabel -> PortHandle
GetPortDetails PortHandle -> ModuleNumber PortNumber
GetPortLabelFromHandle PortHandle -> PortLabel
IsDummyPort PortHandle -> IsDummyPort

AddGroup Handles -> GroupHandle
RemoveGroup GroupHandle
ListGroups -> GroupHandles
ListPortsInGroup GroupHandle -> PortHandles
```

```
AddModule ModuleType
RemoveModule ModuleNumber
ListPortsInModule ModuleNumber -> PortHandles
SetPortComment PortHandle PortComment
GetPortComment PortHandle -> PortComment
GetSessionType -> SessionType
```

Summary Reserves test ports for the current test session. As ports are added, modules are locked for exclusive use by the session.

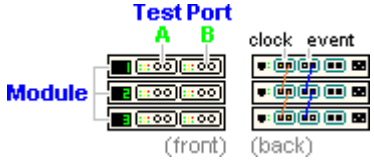
Parameters

FirstModuleNumber long
LastModuleNumber

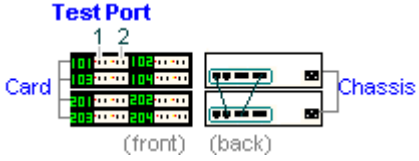
Depends on the method being called:

- GetLastModule: The last test module currently connected to the test system. Use to list all modules that are in the test system.
- GetLockedModules: The first and last test modules, of a range of consecutive modules, that have been locked for exclusive use by this test session.

Modules are assigned incremental numbers, from 1 to LastModuleNumber, according to their order on the Event daisy chain. This number is displayed on the left side of the module's front panel



Test Cards are assigned chassis and card numbers, both starting from 1, according to their order on the Event daisy chain. This number is displayed on the left side of the card's front panel, using the format XXYY, where XX is the chassis number and YY the card number, for example 103 is the bottom left card in chassis 1.



Pass the module and port numbers to the method AddPort, to identify and reserve test ports.

ModuleNumber	long	The number of a particular test module. Modules are assigned incremental numbers, from 1 to LastModuleNumber, according to their order on the Event daisy chain. (See above diagram.) • A test module displays this number on the left side of its front panel. A module number of 0 indicates that module numbers have not yet been assigned.
ModuleType	enum	The type of test module: AGT_CARD_ONEPORT_OMNIBERXM_10G_SONET_1550 AGT_CARD_ONEPORT_OMNIBERXM_10G_SONET_1310 AGT_CARD_TWOPORT_OMNIBERXM_2G5_SONET_1550 AGT_CARD_TWOPORT_OMNIBERXM_2G5_SONET_1310 AGT_CARD_TWOPORT_OMNIBERXM_622M_SONET_1550 AGT_CARD_TWOPORT_OMNIBERXM_622M_SONET_1310
PortNumber	long	The number assigned to a test port. • For each test module, the port labeled “A” is assigned port number “1”, “B” is assigned “2”, etc.
PortType	enum	The type of test port: AGT_PORT_OMNIBERXM_10G_SONET_1550 AGT_PORT_OMNIBERXM_10G_SONET_1310 AGT_PORT_OMNIBERXM_2G5_SONET_1550 AGT_PORT_OMNIBERXM_2G5_SONET_1310 AGT_PORT_OMNIBERXM_622M_SONET_1550 AGT_PORT_OMNIBERXM_622M_SONET_1310
NumberOfPorts	long	The number of test ports on the test module.
IsSynchronized	bool	Indicates whether a test module needs to be synchronized with its neighbors: 1: yes 0: no
IsSupported	bool	Indicates whether a test module is supported by the current session type: 1: yes 0: no Generally, most session types support most module types.
ModuleLimit	long	The maximum number of test modules supported by this type of session. Returns zero if there is no maximum. Generally, the number of modules is not limited by a test session type, only by the hardware platform. Thus, most session types return a value of zero.

ModuleState	enum	The current state of this module: • AGT_MODULE_READY: The module rebooted successfully and is available for use. • AGT_MODULE_LOCKED: The module is locked by a test session. Use GetModuleLock to determine which test session has a lock on the module, and RemovePort to release a session's lock on a port (and thus the module). • AGT_MODULE_REBOOTING: The module is rebooting. • AGT_MODULE_FAILED: The module failed to reboot. When a test session locks a module, the module state changes from READY to LOCKED. When the test session unlocks the module, its state changes from LOCKED to REBOOTING. After the module finishes rebooting, its state changes to either READY (if successful) or FAILED (if it fails to reboot).
SessionHandle	long	A handle to the test session holding a lock on this module. When a session reserves a test port, it locks the entire test module. A session handle of zero indicates an unused module. To get a descriptive text label for the session handle, use the command *AgtGetSessionLabel.*
ModuleNumbers	list<long>	A list of module numbers. These numbers are described above in ModuleNumber.
DummyModuleType	enum	Same as ModuleType above, but is used to add simulated test modules to an offline test session. This may be used to configure tests on a PC that does not have physically connected test modules, or to demo a system with a certain number and types of test modules. To launch an offline test session, use the command *AgtOpenSession* or the object *AgtTestSession*.
DummyModuleNumber	long	Same as ModuleNumber above, but is used to remove simulated test modules in an offline test session. This removes the specified module and all subsequent modules (that is, if you specify module #n and there are N modules, this removes modules #n to N in the simulated chain).
PortHandle	long	A handle to a test port. The handle returned by the method AddPort is used to identify the test port to subsequent commands and methods.

PortHandles	list<long>	A list of port handles. The use depends on the method: • AddPorts: A list of handles to the ports that have been added to a test session. The handles are used to identify test ports to subsequent commands and methods. • RemovePorts: A list of handles to the ports to be removed. • ListPorts: The ports that have been added so far to the current test session. • AddGroup: The ports to define as a group. You may then pass the returned group handle to methods requiring a port handle, to configure a group of ports the same way (that is, calling the method only once). This group definition lasts for the duration of the test session. To create a list of handles to pass to the method, use the Tcl command list (for example, "[list handle1 handle2 ...]"—see this sample code). Note: You cannot keep adding ports to a group, each call creates a new group. • ListPortsInGroup: The ports that are in a group.
IsDummyPort	bool	Indicates whether the test port is being simulated by an _ipl.exe process on the host PC: • 1: Test port is being simulated. • 0: Test port is not being simulated. **Note:** The returned value is always 0 if you are using the test port in an *offline* test session. *Offline* sessions do not actively communicate with actual or simulated test ports.
GroupHandle	long	The handle(s) to a group(s) of ports.
GroupHandles	list<long>	A group consists of a number of port handles. You use a group to configure a number of test ports with the same configuration in a single action.
IsChassisBlade	bool	Set to TRUE if the module is a module in the OmniBER XM chassis).
ChassisNumber	long	The chassis number you wish to lock the event line to.
ChassisSlotNumber	long	Returns the slot number the card is in. Note you can also work this out from the module number itself. For example, card 101 is in slot 1.
PortLabel	string	A label identifying a test port (as used in the GUI).
RequiredModules	list<long>	The list of modules that must be selected in addition to those you actually need for the test.

UnavailableModules	list<long>	The list of modules that you have requested but are not available because they are in use by another session.
SelectedModules	list<long>	The list of modules you would like to use in your test.
PortLabels	list<string>	A list of port labels as described in PortLabel above.
SessionType	enum	The type of test session currently running. One of those listed in AgtOpenSession.
PortComment	string	Provides information on how to configure the port. For example, "Connects to 192.10.3.2". The comment is saved to the configuration file and can be retrieved when restoring a session using AgtTestSession GetSavedPortComment

Details

You reserve test ports for the current test session so that others using the GUI or other API clients do not use the ports and compromise your test results. Once you reserve a test port using AddPort or AddPorts, the entire module becomes locked to the test session. Modules are locked in contiguous blocks. Hence, if you reserve ports 1A and 4A, then modules one through four are locked to the current test session. Ports can be released using RemovePort. To list all modules available in the test system, use the ListModules method.

Details about individual methods:

ListModules: Return the list of modules currently connected to the test system. This includes all OmniBER XM Test Modules.

GetLastModule: Return the number of the last module currently connected to the test system.

GetModuleDescription: Return details of the module with the given module number. Returns the type of the module and the number of ports available in that module. Providing the port count enables clients to discover the number of ports available without having to hardcode details of specific module types.

GetModuleName: Return the host name of a particular OmniBER XM module.

IsModuleSynchronized: Return true if the module must be synchronized with its neighbors.

IsChassisBlade: Return TRUE for all OmniBER XM modules.

GetChassisNumber: For a chassis blade, returns its chassis number.

GetChassisSlotNumber: For a chassis blade, returns its slot number within the chassis.

GetPortType: Return the type of the specified port on the module with the given module number. The ports on each module are numbered from 1 to N, where N is the port count returned by GetModuleDescription.

GetPortLabel: Return the label identifying a specified port (e.g. 1A, 101/1).

GetPortName: Return the name of the specified port.

IsModuleSupported: Returns true if the module is supported by this session type. Generally, most session types support most module types.

ListModuleTypes: Return a list of the module types supported by the module.

SetModuleType: Set the current module type. This method has no effect if the specified module type is the one currently exhibited by the module. The module type can only be set if the module is not currently owned by a test session.

GetModuleLimit: If the number of modules supported by this session type is limited, return the limit. If no limit is defined, return 0. Generally, the number of modules is not limited by a test session type, only by the hardware platform. Thus, most session types return a value of zero.

GetModuleState: Return the current state of the given module.

GetModuleLock: Returns the session ID of the test session holding a lock on the given module number. Returns zero if there is no lock on the given module.

GetLockedModuleList: Returns a list of the modules locked by this session. This replaces the obsoleted GetLockedModules which simply returned a range of module numbers (locked modules are no longer necessarily contiguous).

GetChassisUpstreamLock: Returns the session ID of the test session which has locked this chassis for upstream synchronization. Returns zero if there is no lock.

GetChassisDownstreamLock: Returns the session ID of the test session which has locked this chassis for downstream synchronization. Returns zero if there is no lock.

ListRequiredModules: Given a list of modules to be selected, find any additional modules which must also be selected to preserve synchronization across ports.

ListUnavailableModules: Given a list of modules to be selected, find any modules which are not available to be selected.

AddModule: Add a new simulated module for use in Demo mode. The module will be added to the end of the list. Can only be used in OFFLINE mode.

RemoveModule: Remove the simulated module at the location determined by ModulePosition where the first module is number 1 and the last module, number N. All subsequent modules will be removed as well (i.e., if module 3 is removed, module 3 to module N-1 are removed). Can only be used in OFFLINE mode.

AddPort: Add a test port to the current test session. Return a handle for the port given by the module number and port number. The port number correspond to the lettered ports on the units. For example, Port A is 1, port B is 2. So to add module 5, port B is "AddPort 5 2". This handle can be used anywhere in the remainder of the test system to refer to the test port. When a port is added to the test session, it is automatically locked. To unlock a module, the ports must be removed from the test session. The operation will fail if the module is locked in another session. A port cannot be added to the system while a test is running.

AddPorts: Add a number of test ports to the current test session. AddPorts takes a list of port names (e.g. "1A" or "101/1") instead of a module number and port number.

AddNamedPort: Add a test port to the current test session. Return a handle for the port given by the module name and port name.

RemovePort: Remove a test port from the current test session. When a port is removed all state associated with the port will be lost. If the port is subsequently added again, all associated parameters will have returned to their defaults as if the port was part of a new session. A port cannot be removed from the system while a test is running.

RemovePorts: Remove a number of test ports from the current test session. Faster than removing each port individually, since the event lines must be re-segmented whenever a module is removed from the test session, and this can take a couple of seconds each time.

GetPortDetails: Return the module number and port letter for a given port handle.

IsDummyPort: Return TRUE if the port is a dummy port (an instance of the embedded software running on the host PC).

ListPorts: List ports that are part of the current test session.

AddGroup: Define a new port group consisting of a number of port handles. Used to configure a number of test ports with the same configuration in a single transaction Accepts a list of port handles and returns a group handle.

RemoveGroup: Remove a particular port group definition. Otherwise a group definition will remain for the duration of the test session.

ListGroups: List all groups currently defined.

ListPortsInGroup: List all the port handles that comprise a currently defined group.

Error codes

0: Success.

> 0: Invalid Handle: If PortHandle does not correspond to an active port as assigned by AgtPortSelector; if the ModuleNumber or PortNumber is out of range.

AgtTestSession

Syntax

```
AgtInvoke AgtTestSession Method InParams -> OutParams
```

Methods

The methods marked with * are also provided as commands, for your convenience. (These commands are simply shortcuts to the methods, but do not require the extra AgtInvoke command.) **Note:** The word "Interfaces" used in method names below simply refers to "Objects".

```
OpenSession SessionType SessionMode -> SessionHandle *
CloseSession *
CloseSessionForce
GetHandle -> SessionHandle
GetType -> SessionType
GetMode -> SessionMode
GetContext -> SessionContext
SaveSession FileName *
RestoreSession FileName *
RestoreSessionOnPorts FileName PortLabels
ListSaveableInterfaces -> SaveableObjects
GetSaveableInterfaceDescription Object -> ObjectDescription
ListDependencies Object -> DependentObjects
ListAllDependencies Object -> DependentObjects
ListSavedInterfaces FileName -> SavedObjects *
ListSavedPorts FileName -> PortLabels
GetSavedPortType FileName PortLabel -> PortType
GetSavedPortComment FileName PortLabel -> PortComment
SaveInterfaces FileName ObjectsToSave *
RestoreInterfaces FileName ObjectsToRestore *
ResetSession *
ResetInterfaces ObjectsToReset *
GetNumPorts -> NumPorts
```

The following methods are no longer supported:

```
SetLabel SessionLabel *
GetLabel -> SessionLabel *
```

Use the commands AgtSetSessionLabel, AgtGetSessionLabel instead.

Summary

Manages test sessions.

Note: Most of these methods operate on the current test session. To change the current session, call AgtSetActiveConnection and pass the desired session handle. (The session's handle number is also used as the connection's ID number.)

To list the handles for the active sessions, call AgtListOpenSessions.

Parameters

SessionType	string	The type of test session. For a listing of possible types, see AgtOpenSession.
SessionMode	enum	Whether a session is being used for full testing or configuration only. • • AGT_SESSION_ONLINE: You select this mode if you are doing more than just configuring a test, for example, if you also want to generate traffic and view statistics. This reserves the modules you select on a subsequent call to AgtPortSelector, thus locking out anyone else who might attempt to use the same modules to generate traffic. The test session will actively connect to all selected test ports and download test configurations. Note: The test modules do not need to be connected as you can simulate connected test modules using the system variable AGT_DUMMY_MODULES. • • AGT_SESSION_OFFLINE: Select this mode if you are simply configuring tests. This neither reserves the test modules nor locks out anyone who may want to use the GUI or API to run tests on the same modules. Your test configurations are stored locally on the PC and not downloaded to the test ports. The mode you select affects the software launched, is selected when you first open a session, and cannot be changed afterwards.
SessionHandle	long	If you called OpenSession, this is a handle to the newly opened session. If you called GetHandle, this is the handle to the current test session. To change the current session, call AgtSetActiveConnection and pass the desired session handle. (The session's handle number is also used as the connection's ID number.) To list the handles for the active sessions, call AgtListOpenSessions.

SessionContext	enum	How the current test session was opened. Currently, the only context supported is: • • AGT_SESSION_EXE: The session is running as standalone, detached executable program. Advantages: Multiple GUI clients can access the same test session (that is, its test ports, traffic definitions, real-time statistics). You can exit the GUI without closing its test session or terminating any attached GUI or API clients. This context is no longer supported: • • AGT_SESSION_DLL: The session is running as a DLL hosted by the GUI. When the GUI closes, the test session closes automatically. Advantage: Test sessions locking test modules are not inadvertently left running in the background.
FileName	string	The name of the file used to store the test configuration data. Test configuration files should have the extension .xml. The many different rules for specifying the directory path and file name are detailed for AgtSaveSession.
SaveableObjects	list<string>	A list of the test components (that is, API objects) that may be saved, through the GUI or AgtSaveSession. When you save a test configuration, all the settings associated with these objects are saved.
Object	string	The name of a saveable API object, as returned by the method ListSaveableObjects.
DependentObjects	list<string>	A list of the API objects that are automatically saved when you save the specified Object (through the GUI or AgtSaveSession). The methods are • • ListDependencies: Lists all objects on which the specified object directly depends. • • ListAllDependencies: Lists all objects on which the specified object depends, including all their dependencies, and so on recursively.

SavedObjects	list<string>	A list of the objects that were saved into the specified test configuration file.
ObjectDescription	string	A brief description of the object being saved or restored.
ObjectsToSave ObjectsToRestore ObjectsToReset	list<string>	A list of the objects to save, restore, or reset (that is, to default values).
SessionLabel	string	A descriptive label for the current test session. The default label is • • "SYSTEM" if the session was opened through the API • • user login name (for example, "administrator") if opened through the GUI Use double quotes (") to enclose strings containing spaces.
NumPorts	long	The number of test ports used by the current test session.
PortLabel	string	A port label, for example "1A" or "103/2".
PortLabels	list<string>	A list of port labels. For example, the list "2A 2B 101/1 101/2" might be saved. The session might be restored on the list "101/1 101/2 102/1 102/2". Each label is delimited by a space.
PortType	string	The type of port, as returned by AgtPortSelector GetPortType method.
PortComment	string	A comment, associated with a port, providing information on configuring the port. For example "Connect to interface X on omniber xm Y".

Details You interact with the test system through a test session, which is simply an instantiation of the test system software. Test sessions can be initiated by launching the graphical user interface or by calling the command AgtOpenSession.

You connect to a running test session using AgtConnect and disconnect using AgtDisconnect. You can set up multiple connections to different test sessions but there is always only one active connection.

AgtCloseSession closes a currently open test session, and all connections to that session. The test session associated with the graphical user interface is automatically closed when the graphical user interface exits and should not be closed by a script.

Details about individual methods:

OpenSession: Opens a session of the given type. The available types and modes described for the command AgtOpenSession. The mode controls whether the session is connected to actual modules (ONLINE) or just storing the configuration locally (OFFLINE).
CloseSession: Closes this test session but fails if a GUI client is attached to it or a test is starting.

CloseSessionForce: Closes this test session, regardless of these conditions.

GetHandle: Returns a handle for this session. Returns zero if the session is not currently open. **GetType:** Returns the type of this session. **GetMode:** Returns the current mode of this session. **GetContext:** Returns the context of the current test session.

ListSaveableInterfaces: Returns a list of the interfaces in the test session that can be saved.
ListDependencies: For a given interface name, returns a list of all of the interfaces on which the interface depends directly.

ListAllDependencies: For a given interface name, returns a list of all of the (saveable) interfaces on which the interface depends, including all of their dependencies, and so on recursively.

GetSaveableInterfaceDescription: For a given interface name, returns a string description of the interface. **SaveInterfaces:** Saves the requested list of interfaces in the test session All dependents will be automatically saved.

SaveSession: Saves all saveable state in the test session to the requested filename. **RestoreSession:** Restores the state of the test session from the supplied file. For each interface whose persistent state exists in the given file, the session will reset its state to its default prior to restoring the state from file. The session will be restored on the original ports. Use

RestoreSessionOnPorts to restore on a different set of ports. NOTE: the number of ports must equal the original list. **ResetSession:** Resets the test session to its default state.

ListSavedInterfaces: Returns a list of the interfaces saved in the supplied configuration file Allows the contents of the file to be summarized before restoring. **RestoreInterfaces:** Restores the state of the test session from the supplied file The client lists the interfaces to restore For each interface whose persistent state exists in the given file, the session will reset its state to its default prior to restoring the state from file. **ResetInterfaces:** Resets a subset of the test session configuration The client lists the interfaces to reset For each interface the session will reset its state to its default prior to restoring the state from file.

GetNumPorts: Returns the number of ports used by this test session.

Error codes

0 Success.

1 Bad argument.

AgtXmSettings

This interface is common to SONET and SDH.

Summary

This interface is used to configure the current signal format. Also sets the SONET/SDH scrambler on/off. All the functions are available in Terminal mode but only the Receiver functions are available in Thru Mode.

Syntax

```
AgtInvoke AgtXmSettings Method InParams -> OutParams
```

Methods

```
SetTxSignalStandard PortHandle SignalStandardMode
GetTxSignalStandard PortHandle → SignalStandardMode

SetTxLineRate PortHandle TxLineRate
GetTxLineRate PortHandle → TxLineRate

SetTransmitterMode PortHandle TransmitterMode
GetTransmitterMode PortHandle → TransmitterMode

SetScramblerState PortHandle ScramblerState
GetScramblerState PortHandle → ScramblerState

SetTxLineRateOffset PortHandle LineRateOffset
GetTxLineRateOffset PortHandle → LineRateOffset
GetValidTxLineRateOffsetRange PortHandle → MinOffset
MaxOffset
```

Note: *Only an information function. No "Set" functionality.*

Receiver Functions

```
SetRxSignalStandard PortHandle SignalStandardMode
GetRxSignalStandard PortHandle → SignalStandardMode

SetRxLineRate PortHandle RxLineRate
GetRxLineRate PortHandle à RxLineRate

SetDescramblerState PortHandle DescramblerState
GetDescramblerState PortHandle -> DescramblerState

SetMeasurementAnalysisType PortHandle AnalysisType
GetMeasurementAnalysisType PortHandle -> AnalysisType

SetEnhancedRdipModeForAnalysis PortHandle State
SetEnhancedRdipModeForAnalysis PortHandle -> State

SetMonitorMode PortHandle MonitorMode
GetMonitorMode PortHandle -> MonitorMode
```

Note: *Only applies in Thru mode*

Parameters

SignalStandardMode	Enum	EAgtXmSignalStandard AGT_XM_SIGNAL_STANDARD_SONET AGT_XM_SIGNAL_STANDARD _SDH
Rx/TxLineRate	Enum	EAgtXmLIneRate AGT_XM_LINE_RATE_10G_SONET AGT_XM_LINE_RATE_10G_SDH AGT_XM_LINE_RATE_2G5 AGT_XM_LINE_RATE_622M AGT_XM_LINE_RATE_155M
LineRateOffset	long	The offset added to each timestamp, in parts per million (ppm). Default is 0, valid range varies per blade
MinOffset	long	The minimum valid LineRate offset value (in ppm) for this port
MaxOffset	long	The maximum valid LineRate offset value (in ppm) for this port
PortHandle	long	A handle to a test port, as returned by AgtPortSelector
ScramblerState	Bool	Indicates whether the test port is currently scrambling its transmitted SONET/SDH frames:· • 1: Scrambler is on. • 0: Scrambler is off.
DescramblerState	Bool	Indicates whether the test port is currently descrambling SONET/SDH frames from the SUT: • 1: Descrambler is on. • 0: Descrambler is off.
MonitorMode	Enum EAgtXmMonitorMode	AGT_XM_MONITOR_TRANSPARENT AGT_XM_MONITOR_INTRUSIVE
AnalysisType	Enum	AGT_XM_ANALYSIS_TYPE_G828 AGT_XM_ANALYSIS_TYPE_GR253 AGT_XM_ANALYSIS_TYPE_NONE

Details The SDH signal is scrambled according to G.707. The scrambler is frame-synchronous and uses an X7+X6+1 polynomial, XORed with the data.

For setting LineRate, the valid options are dependent upon the blade being used. In particular 10G blade doesn't support any other rate, while 2.5G blade supports 622 Mbps & 155 Mbps.

Error codes

0	Success
>0	Invalid Handle: If PortHandle does not correspond to an active port as assigned by AgtPortSelector.

AgtXmSonetTransportOverhead

Summary

This interface gets the value of section and line (Transport) overhead bytes being used in transmitted and received SONET frames.
Sets the values of certain transmitted bytes contained in the first STS-3 channel, except for B1, B2, J0 and H1 - H3. All the functions are available in Terminal mode but only the Receiver functions are available in Thru Mode.

Syntax

```
AgtInvoke AgtXmSonetTransportOverhead Method InParams ->
OutParams
```

Methods

```
GetTxOverheadByteMode PortHandle Sts1Pos Byte -> ByteMode
```

Gets the Mode of the overhead byte. The Mode indicates whether the byte can be edited, or whether its value is fixed.

```
SetAllTxOverheadBytesToDefaultValue PortHandle
SetTxOverheadByteToDefaultValue PortHandle Sts1Pos Byte

SetTxOverheadByte PortHandle Sts3Pos Byte Sts1Col Value
GetTxOverheadByte PortHandle Sts3Pos Byte Sts1Col -> Value

SetSts3TxOverheadBytes PortHandle Sts3Pos
OverheadBytesSnapshot
GetSts3TxOverheadBytes PortHandle Sts3Pos ->
OverheadBytesSnapshot

SetTxAps PortHandle K1 K2
GetTxAps PortHandle -> K1 K2

SetTxSectionTraceMessageToDefault PortHandle

Note:Thru Mode has only the following functions:
SetTxSectionTraceLength PortHandle TraceLength
GetTxSectionTraceLength PortHandle -> TraceLength

SetTxSectionTraceMessage PortHandle SectionTrace
GetTxSectionTraceMessage PortHandle -> SectionTrace
```

Receiver Functions:

```
GetRxOverheadByte PortHandle Byte Sts1Col -> Sts3Pos Value
GetSts3RxOverheadBytes PortHandle -> Sts3Pos
OverheadBytesSnapshot
Returns data from the currently selected STS3.
```

```
GetRxAps PortHandle -> K1 K2

GetRxSectionTraceMessage PortHandle -> SectionTraceMessage

GetRxSectionTraceLength PortHandle -> Length

SetCurrentRxSts3 PortHandle Sts3Pos
GetCurrentTxSts3 PortHandle -> Sts3Pos
```

Parameters

PortHandle	long	A handle to a test port, as returned by AgtPortSelector
Byte	Enum EAgtXmSonet Transport OverheadByte	An overhead byte in SONET/SDH frames: For descriptions of the overhead bytes, see this quick reference on SONET/SDH frame header. You can get the value of any byte in transmitted and received frames. The non-editable bytes in transmitted frames are set to either a fixed or automatically calculated (correct) value. • AGT_XM_SONET_A1 • AGT_XM_SONET_A2 • AGT_XM_SONET_J0 • AGT_XM_SONET_Z0 • AGT_XM_SONET_B1 • AGT_XM_SONET_E1 • AGT_XM_SONET_F1 • AGT_XM_SONET_D1 • AGT_XM_SONET_D2 • AGT_XM_SONET_D3 • AGT_XM_SONET_H1 • AGT_XM_SONET_H2 • AGT_XM_SONET_H3 • AGT_XM_SONET_B2 • AGT_XM_SONET_K1 • AGT_XM_SONET_K2 • AGT_XM_SONET_D4 • AGT_XM_SONET_D5 • AGT_XM_SONET_D6 • AGT_XM_SONET_D7 • AGT_XM_SONET_D8 • AGT_XM_SONET_D9 • AGT_XM_SONET_D10 • AGT_XM_SONET_D11 • AGT_XM_SONET_D12

		• AGT_XM_SONET_S1 • AGT_XM_SONET_Z1 • AGT_XM_SONET_Z2 • AGT_XM_SONET_M1 • AGT_XM_SONET_M0 • AGT_XM_SONET_E2
ByteMode	Enum EAgtXmSonet TransportOverhead ByteMode	How a particular byte in the SONET/SDH overhead is set by the tester. • AGT_XM_EDITABLE_BYTE: This byte has a default, but can be user-edited. Available for all the bytes except B1, B2, J0 and H1, H2, H3 bytes. J0 excluded from STS3-1 only, it becomes Z if other STS3s and can be edited. • AGT_XM_FIXED_BYTE: This byte is not user editable. Includes B1, B2, J0 and H1, H2, H3 bytes. J0 only appears in STS3-1
Sts1Col	Long	Indicates which STS1 column within the current STS3 should be used. Valid range 1-3.
Value	unsigned char	8-bit integer (unsigned char) specifying the value of the overhead byte
K1,K2	unsigned char	Automatic Protection Switching, K1 and K2 bytes. Enables downstream line-terminating equipment to initiate protection switching upon detection of line defects, by switching to standby systems (in linear APS, bidirectional line-switched rings). Default is Not Used (0x0000)
TraceLength	Enum	EAgtXmTraceLength • AGT_XM_SECTION_TRACE_16_BYTES • AGT_XM_SECTION_TRACE_64_BYTES
SectionTrace	String	User defined string. Could be 15 or 62 bytes long depending on the current selected setting of trace mode.
Sts3PosF	Long	Indicates the STS3 header that is being viewed/controlled. Valid range depends on the line speed. 10G = [1-64]; 2G5 = [1-16]; 622M = [1-4]; 155M = [1]

Length	Enum	Length of the rceieved Section trace message. Possible values are 15, 62 or 64.
OverheadBytesSnapshot	Array	An array [1 X 81] of Overhead bytes. Same order as the frame transmission. On transmission; user specified values for B1, B2, J0 and H1, H2, H3 bytes will be ignored

AgtXmSdhSectionOverhead

Summary

This interface gets the value of section and line (Transport) overhead bytes being used in transmitted and received SDH frames. Sets the values of certain transmitted bytes contained in the first STM-1 channel, except for B1, B2, J0 and H1 - H3.

Syntax

```
AgtInvoke AgtXmSdhTransportOverhead Method InParams ->
OutParams
```

Methods

```
GetTxOverheadByteMode PortHandle Stm1Pos Byte Stm0Col ->
ByteMode
Gets the Mode of the overhead byte. The Mode indicates
whether the byte can be edited, or whether its value is
fixed.

SetAllTxOverheadBytesToDefault PortHandle
SetTxOverheadByteToDefault PortHandle Stm1Pos Byte Stm0Col

SetTxOverheadByte PortHandle Stm1Pos Byte Stm0Col Value
GetTxOverheadByte PortHandle Stm1Pos Byte Stm0Col -> Value

SetStm1TxOverheadBytes PortHandle Stm1Pos
OverheadBytesSnapshot
GetStm1TxOverheadBytes PortHandle Stm1Pos ->
OverheadBytesSnapshot

SetTxAps PortHandle K1 K2
GetTxAps PortHandle -> K1 K2

SetTxSectionTraceMessageToDefault PortHandle

SetTxSectionTraceLength PortHandle TraceLength
GetTxSectionTraceLength PortHandle -> TraceLength

SetTxSectionTraceMessage PortHandle SectionTrace
GetTxSectionTraceMessage PortHandle -> SectionTrace

Note: The following are Thru mode functions.

GetRxOverheadByte PortHandle Sts1Pos Byte ->Stm1Pos Value
GetStm1RxOverheadBytes PortHandle Stm1Pos
OverheadBytesSnapshot
GetRxAps PortHandle -> K1 K2

GetRxSectionTraceMessage PortHandle -> SectionTraceMessage
GetRxSectionTraceLength PortHandle -> Length

SetCurrentRxStm1 PortHandle Stm1Pos
GetCurrentRxStm1 PortHandle -> Stm1Pos
```

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector
Byte	Enum	An overhead byte in SDH frames: For descriptions of the overhead bytes, see this quick reference on SDH frame header. You may get the value of any byte in received frames. The non- editable bytes in transmitted frames are set to either a fixed or automatically calculated (correct) value. • AGT_XM_SDH_A1 • AGT_XM_SDH_A2 • AGT_XM_SDH_J0 • AGT_XM_SDH_Z0 • AGT_XM_SDH_B1 • AGT_XM_SDH_E1 • AGT_XM_SDH_F1 • AGT_XM_SDH_D1, • AGT_XM_SDH_D2. • AGT_XM_SDH_D3 • AGT_XM_SDH_H1, • AGT_XM_SDH_H2 • AGT_XM_SDH_H3 • AGT_XM_SDH_B2 • AGT_XM_SDH_K1 • AGT_XM_SDH_K2 • AGT_XM_SDH_D4, • AGT_XM_SDH_D5 • AGT_XM_SDH_D6, • AGT_XM_SDH_D7 • AGT_XM_SDH_D8, • AGT_XM_SDH_D9 • AGT_XM_SDH_D10, • AGT_XM_SDH_D11 • AGT_XM_SDH_D12 • AGT_XM_SDH_S1 • AGT_XM_SDH_Z1 • AGT_XM_SDH_Z2 • AGT_XM_SDH_M1 • AGT_XM_SDH_M0 • AGT_XM_SDH_E2

ByteMode	Enum	How a particular byte in the SDH overhead is set by the tester. • AGT_XM_EDITABLE_BYTE: This byte has a default, but can be user-edited. Available for all the bytes except B1, B2, J0 and H1, H2, H3 bytes. • AGT_XM_FIXED_BYTE: This byte is not user editable. Includes B1, B2, J0 and H1, H2, H3 bytes. J0 only appears in STS3-1
Stm0Col	Long	Indicates which STM0 column within the current STM1 should be used. Valid range 1-3.
Value	unsigned char	8-bit integer (unsigned char) specifying the value of the overhead byte.
K1, K2	unsigned char	Automatic Protection Switching, K1 and K2 bytes. Enables downstream line-terminating equipment to initiate protection switching upon detection of line defects, by switching to standby systems (in linear APS, bidirectional line-switched rings). Default is Not Used (0x0000).
TraceLength	Enum	• AGT_XM_TRACE_16_BYTES • AGT_XM_TRACE_64_BYTES •
SectionTrace	String	User defined string. Could be 15 or 62 bytes long depending on the current selected setting of trace mode.
Stm1Pos	Long	Indicates the STM1 header that is being viewed/controlled. Valid range depends on the line speed. 10G = [1-64]; 2G5 = [1-16]; 622M = [1-4]; 155M = [1]
Length	Long	Length of the received Section trace message. Possible values are 15, 62 or 64.
OverheadBytesSnapshot	Array	An array [1 X 81] of Overhead bytes. Same format as the frame transmission. User specified values for B1, B2, J0 and H1, H2, H3 bytes will be ignored.

AgtXmSonetError

Summary This interface is used to controlling the error injection in the transmitted signal. Only B1, B2 and B3 errors could be injected in Thru Mode.

Syntax

```
AgtInvoke AgtXmSonetError Method InParams -> OutParams
```

Methods

```
SetTxError PortHandle Error Type
GetTxError PortHandle -> ErrorType

GetREILErrorMode PortHandle ->REILErrorMode
SetREILErrorMode PortHandle REILErrorMode

GetErrorRateRange PortHandle ErrorRateType
->MinErrorRateBase ->MinErrorRatePower  ->MaxErrorRateBase
->MaxErrorRatePower

SetErrorRate PortHandle ErrorRateType ErrorRateBase
ErrorRatePower

GetErrorRate PortHandle ErrorRateType ->ErrorRateBase
->ErrorRatePower

GetStrErrorRate PortHandle SonetRateType ->ErrorRate

AddSingleError PortHandle

ErrorRateOn PortHandle
ErrorRateOff PortHandle
IsErrorRateOn PortHandle -> State (Bool)

ListValidErrorTypes PortHandle -> ErrorType[]

SetServiceDisruptionGuardTime PortHandle GuardTime
GetServiceDisruptionGuardTime PortHandle -> GuardTime
SetServiceDisruptionGuardTimeToDefault PortHandle
GetMaxServiceDisruptionTime PortHandle -> DisruptionTime

SetTxErrorMode PortHandle ErrorMode
GetTxErrorMode PortHandle -> ErrorMode

ErrorRateOnAllPorts
ErrorRateOffAllPorts
AddSingleErrorAllPorts
```

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector.	
ErrorRateBase	float	1.00-9.99	
ErrorRatePower	EAgtXmError RatePower	AGT_XM_ERROR_RATE_1E-3 AGT_XM_ERROR_RATE_1E-4 AGT_XM_ERROR_RATE_1E-5 AGT_XM_ERROR_RATE_1E-6 AGT_XM_ERROR_RATE_1E-7 AGT_XM_ERROR_RATE_1E-8 AGT_XM_ERROR_RATE_1E-9 AGT_XM_ERROR_RATE_1E-10	
Error Rate	String	Logical Concatenation of *ErrorRateBase* and *ErrorRatePower* for example "3.4 * 1E-5"	
Error Type	EAgtXmSonet ErrorType EagtXmSdhErr orType	**EAgtXmSonetError** AGT_XM_SONET_B1_ERROR AGT_XM_SONET_B2_ERROR AGT_XM_SONET_REIL_ERROR AGT_XM_SONET_B3_ERROR AGT_XM_SONET_REIP_ERROR AGT_XM_SONET_BIT_ERROR AGT_XM_SONET_REIV_ERROR AGT_XM_SONET_BIP_ERROR	**EAgtXmSdhError** AGT_XM_SDH_B1_ERROR AGT_XM_SDH_B2_ERROR AGT_XM_SDH_MSREI_ERROR AGT_XM_SDH_B3_ERROR AGT_XM_SDH_BIT_ERROR AGT_XM_SDH_HPREI_ERROR AGT_XM_SDH_LPREI_ERROR AGT_XM_SDH_TUBIP_ERROR
REILErrorMode	EAgtXmREIL ErrorMode	AGT_REIL_M1_MO_MODE AGT_REIL_M1_ONLY_MODE	

Error Rate Type	EAgtXmSonet ErrorRate EAgtXmSdh ErrorRate	**Sonet** AGT_XM_SONET_LINE AGT_XM_SONET_STS1 AGT_ XM_SONET_STS3c AGT_ XM_SONET_STS6c AGT_ XM_SONET_STS9c AGT_ XM_SONET_STS12c AGT_ XM_SONET_STS24c AGT_ XM_SONET_STS48c AGT_ XM_SONET_STS192c AGT_XM_SONET_VT1_5 AGT_XM_SONET_VT2 AGT_XM_SONET_TU3	**SDH** AGT_XM_SDH_LINE AGT_XM_SDH_AU3 AGT_XM_SDH_AU4 AGT_XM_SDH_AU4_2c AGT_XM_SDH_AU4_3c AGT_XM_SDH_AU4_4c AGT_XM_SDH_AU4_8c AGT_XM_SDH_AU4_16c AGT_XM_SDH_TU11 AGT_XM_SDH_TU12 AGT_XM_SDH_TU3
GuardTime	Long	The guard time used to define the end of a burst is user configurable between 100ms and 1600ms in 1ms steps. This will be accurate to +/- 0.5ms. The default guard time is 200ms.	
ErrorMode	EAgtXmError BurstMode	AGT_XM_ERROR_MODE_MANUAL AGT_XM_ERROR_MODE_TIMED	
ErrorRateState	Bool	• 1: Error is on. • 0: Error is off.	
DisruptionTime	Double	The maximum service disruption time that can be measured. (Note this is not the maximum service disruption time that has occurred on the port)	

Error codes

0	Success
>0	Invalid Handle: If *PortHandle* does not correspond to an active port as assigned by *AgtPortSelector*.

Details

Timed burst mode not supported for VT/TU errors.

AgtXmSdhError

See the list of commands given for AgtXmSonetError.

AgtXmSonetAlarm

Summary

This interface is used to controlling the alarm injection in the transmitted signal. Only LOS, LOF, LOP, AIS-P, alarms can be injected in Thru Mode. AIS-V, LOP-V, RDI-V, UNEQ-V, and RFI-V are only available in VT mode.

Syntax

```
AgtInvoke AgtXmSonetAlarm Method InParms -> OutParms
```

Methods

```
SetTxAlarm PortHandle AlarmType
GetTxAlarm PortHandle -> AlarmType

SetAlarmValue PortHandle AlarmType Value (only for PDI-P &
RDI-P)
GetAlarmValue PortHandle AlarmType -> Value (only for PDI-P
& RDI-P)
SetAlarmValueToDefault PortHandle AlarmType (only for PDI-P
& RDI-P)

EnhancedRdipModeOn PortHandle
EnhancedRdipModeOff PortHandle
IsEnhancedRdipModeOn PortHandle -> State (BOOL)

AlarmOn PortHandle
AlarmOff PortHandle
IsAlarmOn PortHandle
SetAlarmValueToDefault PortHandle AlarmType
TransmitAlarm PortHandle
ListValidAlarmTypes PortHandle -> AlarmType[]

Note: Returned list is valid for the current operating mode
for the port.  The list will be empty if no alarms are valid
in the current mode.

SetTxAlarmMode PortHandle AlarmMode
GetTxAlarmMode PortHandle -> AlarmMode

AlarmOnAllPorts
AlarmOffAllPorts
TransmitAlarmAllPorts

The 'SetTxAlarmMode' method may return E_AGT_RESOURCE_IN_USE
if alarms are switched on when the user attempts to change
the operating mode.  The 'SetTxAlarmMode' method may return
E_AGT_INVALID_OPERATION if attempting to set 'pulsed' mode
when LOS alarm type selected
```

Parameters

PortHandle	long	A handle to a port, as returned by AgtPortSelector.	
Value	long	PDI-P Any 8-bit value Enhanced RDIP 2, 5 and 6. Non-Enhanced RDIP 4 and 7.	
AlarmType	EAgtXmSonetAlarmType/ EAgtXmSdhAlarmType		
		AGT_XM_SONET_ALARM_NONE	AGT_XM_SDH_ALARM_NONE
		AGT_XM_SONET_LOS	AGT_XM_SDH_LOS
		AGT_XM_SONET_SEF	AGT_XM_SDH_OOF
		AGT_XM_SONET_LOF	AGT_XM_SDH_LOF
		AGT_XM_SONET_AISL	AGT_XM_SDH_MSAIS
		AGT_XM_SONET_RDIL	AGT_XM_SDH_MSRDI
		AGT_XM_SONET_AISP	AGT_XM_SDH_AUAIS
		AGT_XM_SONET_LOPP	AGT_XM_SDH_AULOP
		AGT_XM_SONET_RDIP	AGT_XM_SDH_HPRDI
		AGT_XM_SONET_UNEQP	AGT_XM_SDH_HPUNEQ
		AGT_XM_SONET_PSL	AGT_XM_SDH_PSL
		AGT_XM_SONET_PDIP	AGT_XM_SDH_PDIP
	VT/TU Commands	AGT_XM_SONET_H4_LOM	AGT_XM_SDH_H4_LOM
		AGT_XM_SONET_AISV	AGT_XM_SDH_TUAIS
		AGT_XM_SONET_LOPV	AGT_XM_SDH_TULOP
		AGT_XM_SONET_RDIV	AGT_XM_SDH_TURDI
		AGT_XM_SONET_UNEQV	AGT_XM_SDH_TUUNEQ
		AGT_XM_SONET_RFIV	AGT_XM_SDH_TURFI
		AGT_XM_PDH_AIS	AGT_XM_PDH_AIS
		AGT_XM_PDH_LOF	AGT_XM_PDH_LOF
TransmitterMode	Enum	AGT_TX_TERMINAL_MODE AGT_TX_THRU_MODE	
AlarmMode	EAgtXmAlarmBurstMode	AGT_XM_ALARM_MODE_MANUAL AGT_XM_ALARM_MODE_PULSED AGT_XM_ALARM_MODE_TIMED	

Details You can only generate a single SONET/SDH alarm type for each port within the test session. SEF/OOF is a one-shot alarm that is injected by calling TransmitAlarm. All VT/TU and PDH alarms do not support pulsed or timed burst modes.

AgtXmSdhAlarm

See the list of commands given for AgtXmSonetAlarm.

AgtXmStatus

Summary

This interface checks if any SONET/SDH alarms or errors were detected in the last sampling interval, and is available in both Terminal and Thru Mode.

The interface is common for both SONET/SDH. Same bits are mapped to analogous SONET/SDH errors.

Syntax

```
AgtInvoke AgtXmStatus Method InParams -> OutParams
```

Methods

```
GetPortSummaryStatus -> StatusRegister
GetPortStatus PortHandle -> StatusRegister
GetPathSummaryStatus PortHandle -> StatusRegister
GetPathStatus PortHandle PathPos -> StatusRegister

GetLoSummaryStatus PortHandle PathPos -> StatusRegister
GetLoStatus PortHandle PathPos LoNumber -> StatusRegister

GetPortSummaryHistory PortHandle -> HistoryStatusRegister
GetPortHistory PortHandle -> HistoryStatusRegister
GetPathSummaryHistory PortHandle -> HistoryStatusRegister
GetPathHistory PortHandle PathPos -> HistoryStatusRegister

GetLoSummaryHistory PortHandle PathPos ->
HistoryStatusRegister
GetLoHistory PortHandle PathPos LoNumber ->
HistoryStatusRegister
ClearHistory PortHandle

GetPortSummaryEventRegister -> EventRegister
GetPortEventRegister PortHandle -> EventRegister
GetPathSummaryEventRegister PortHandle -> EventRegister
GetPathEventRegister PortHandle PathPos -> EventRegister
GetLoSummaryEventRegister PortHandle PathPos ->
EventRegister
GetLoEventRegister PortHandle PathPos LoNumber ->
EventRegister

SetPortSummaryEventEnableRegister EnableRegister
GetPortSummaryEventEnableRegister -> EnableRegister
SetPortEventEnableRegister PortHandle EnableRegister
GetPortEventEnableRegister PortHandle -> EnableRegister
SetPathSummaryEventEnableRegister PortHandle EnableRegister
GetPathSummaryEventEnableRegister PortHandle ->
EnableRegister
SetPathEventEnableRegister PortHandle PathPos EnableRegister
GetPathEventEnableRegister PortHandle PathPos ->
EnableRegister
SetLoSummaryEventEnableRegister PortHandle PathPos
EnableRegister
GetLoSummaryEventEnableRegister PortHandle PathPos ->
EnableRegister
```

```
SetLoEventEnableRegister PortHandle PathPos LoNumber
EnableRegister
GetLoEventEnableRegister PortHandle PathPos LoNumber ->
EnableRegister

SetPortSummaryNegativeTransitionFilter TransitionFilter
GetPortSummaryNegativeTransitionFilter -> TransitionFilter
SetPortNegativeTransitionFilter PortHandle TransitionFilter
GetPortNegativeTransitionFilter PortHandle ->
TransitionFilter
SetPathSummaryNegativeTransitionFilter PortHandle
TransitionFilter
GetPathSummaryNegativeTransitionFilter PortHandle ->
TransitionFilter
SetPathNegativeTransitionFilter PortHandle PathPos
TransitionFilter
GetPathNegativeTransitionFilter PortHandle PathPos ->
TransitionFilter
SetLoSummaryNegativeTransitionFilter PortHandle PathPos
TransitionFilter
GetLoSummaryNegativeTransitionFilter PortHandle PathPos ->
TransitionFilter
SetLoNegativeTransitionFilter PortHandle PathPos LoNumber
TransitionFilter
GetLoNegativeTransitionFilter PortHandle PathPos LoNumber ->
TransitionFilter
SetPortSummaryPositiveTransitionFilter TransitionFilter
GetPortSummaryPositiveTransitionFilter -> TransitionFilter
SetPortPositiveTransitionFilter PortHandle TransitionFilter
GetPortPositiveTransitionFilter PortHandle ->
TransitionFilter
SetPathSummaryPositiveTransitionFilter PortHandle
TransitionFilter
GetPathSummaryPositiveTransitionFilter PortHandle ->
TransitionFilter
SetPathPositiveTransitionFilter PortHandle PathPos
TransitionFilter
GetPathPositiveTransitionFilter PortHandl PathPos ->
TransitionFilter
SetLoSummaryPositiveTransitionFilter PortHandle PathPos
TransitionFilter
GetLoSummaryPositiveTransitionFilter PortHandle PathPos ->
TransitionFilter
SetLoPositiveTransitionFilter PortHandle PathPos LoNumber
TransitionFilter
GetLoPositiveTransitionFilter PortHandle PathPos LoNumber ->
TransitionFilter

GetSummaryStatus PortHandle -> Status
GetSummaryHistory PortHandle -> History

GetAllLoStatus PortHandle -> Status[]
GetAllLoHistory PortHandle -> History[]
```

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector.
StatusRegister	Long	SONET/SDH status bits Only 16 least significant bits are used. See below for explanation of bits.This register maintains the status of the errors, alarms or events during the last sample period. There is no latching of conditions in this register, it is updated in real time.
HistoryStatusRegister	Long	SONET/SDH Status History bits Only 16 least significant bits are used. See below for explanation of bits.This returns the Status over the period since the last ClearHistory function was called.
EventRegister	Long	SONET/SDH Event Register bits Only 16 least significant bits are used. See below for explanation of bits.Latches the transient states that occur in the Condition Register as specified by the Transition Filters. The act of reading these registers resets their contents to all 0's. These bits contribute towards the summary message from the Data Structure.
EnableRegister	Long	SONET/SDH Event Enable Register bits Only 16 least significant bits are used. See below for explanation of bits.Masks the Event Register, determining which of its bits set the summary bit in the Summary Message.
TransitionFilter	Long	SONET/SDH Event Transition Filter bits Only 16 least significant bits are used. See below for explanation of bits.Determines whether positive or negative or both transitions in the Condition Register set the Event Register

Path Register

DB0	UNEQ	Indicates the specified path is unequipped
DB1	REI-P	Indicates Path REI on the specified path
DB2	B3	Indicates B3 Errors on the specified path
DB3	LOP	Indicates LOP on the specified path.
DB4	PDI-P	Indicates PDI-P on the specified path.
DB5	AIS-P	Indicates Path AIS on the specified path
DB6	PNTR	Indicates Pointer Adjust on the specified path
DB7	SDIR	Indicates Service Disruption on the specified path
DB8	not used	
DB9	not used	

DB10	RDI-P	Indicates Path RDI on the specified path.
DB11	BIT-ER	Indicates Bit Errors on the specified path.
DB12	not used	
DB13	PSL	Indicates Pattern Sync Loss on the specified path.
DB14	SUMM-V	VT Event Summary - indicates an event or events are signalled in at least one of the reeceived VT's for the specified path.
DB15	not used	And must never be used

The layout for the Path Summary Register set is identical to the above Path Register set layout.

Port Register

DB0	LOS	Indicates LOS at the specified port
DB1	LOP	Indicates LOF at the specified port
DB2	SEF	Indicates SEF at the specified port
DB3	B1	Indicates B1 errors on the specified port.
DB4	AIS-L	Indicates Line AIS on the specified port.
DB5	B2	Indicates B2 errors on the specified port
DB6	REI-L	Indicates Line REI-L on the specified port
DB7	not used	
DB8	not used	
DB9	RDI-L	Indicates Line RDI on the specified port
DB10	not used	
DB11	not used	
DB12	not used	
DB13	not used	
DB14	not used	
DB15	not used	And must never be used

The layout for the Port Summary Register is:

DB0 - SUMM-P - Set if any path register bit is set for this port.
DB1 to DB15 - Not used.

VT/TU Register

DB0	UNEQ-V	Indicates the specified VT is unequipped
DB1	REI-V	Indicates REI-V on the specified VT
DB2	BIP	Indicates BIP on the specified VT
DB3	LOP-V	Indicates LOP-V on the specified VT
DB4	RFI-V	Indicates RFI-V on the specified VT
DB5	AIS-V	Indicates AIS-V on the specified VT
DB6	PNTR	Indicates Pointer Adjust on the specified VT
DB7	SDIR	Indicates Service Disruption on the specified VT
DB8		

DB9	Not Used	-
DB10	RDI-V	Indicates RDI-V on the specified VT
DB11	BIT-ER	Indicates Bit Errors on the specified VT
DB12	Not Used	-
DB13	PSL	Indicates Pattern Sync Loss on the specified VT
DB14	Not Used	-
DB15	Not Used	Not to be used

AgtXmSonetPathOverhead

Summary

This interface gets the value of Path overhead bytes being used in transmitted and received SONET/SDH frames. Each of the POH bytes in each channel, except for J1& B3 may be individually set.

Syntax

`AgtInvoke AgtXmSonetPathOverhead` *Method InParams -> OutParams*

Methods

`GetTxPathOverheadByteMode PortHandle Sts1Pos Byte -> ByteMode`
Gets the Mode of the overhead byte. The Mode indicates whether the byte can be edited, or whether its value is fixed.

`SetTxByteToDefaultValue PortHandle Sts1Pos Byte`
Returns an error if an attempt to change a fixed POH value.

`SetAllTxBytesToDefaultValue PortHandle Sts1Pos`
Sets all the bytes of the POH to their default values. Note that calling this function from the SONET and SDH interfaces loads different values into the C2 byte.

`SetAllChannelsAllTxBytesToDefaultValue portHandle`
Sets all the bytes of the POH in all the channels to their default values. Note that calling this function from the SONET and SDH interfaces loads different values into the C2 byte.

`SetTxPathOverheadByte PortHandle Sts1Pos Byte Value`
Changes an individual byte within the POH for a particular channel. Returns an error if an attempt to change a fixed POH value.

`GetTxPathOverheadByte PortHandle Sts1Pos Byte -> Value`
Returns an error if an attempt to change a fixed POH value.

`SetTxPathOverheadHeader PortHandle Sts1Pos PathOverheadHeader`
User specified values for J1 & B3 bytes will be ignored.

`GetTxPathOverheadHeader PortHandle Sts1Pos -> PathOverheadHeader`
J1/B3 bytes are set to zero.

`SetTxPathTraceMessageLength PortHandle Sts1Pos TraceLength`
The system defaults to the 64 byte message format at initialisation. When the user changes the length of the PathTraceMessage the previously set value for the new length is restored. (If no previous value exists the default message will be used.)

`GetTxPathTraceMessageLength PortHandle Sts1Pos -> TraceLength`

SetTxPathTraceMessage PortHandle Sts1Pos PathTraceMessage
If the user has not defined a message, then the default message (of the appropriate length) will be used.

SetTxPathTraceMessageToDefault PortHandle Sts1Pos
Sets the transmit path trace message to the default. This is applied to the currently configured length of message for the channel in question. The other length of message is left unchanged.

GetTxPathTraceMessage PortHandle Sts1Pos -> PathTraceMessage
Returns the actual message being sent which means that if the default message mode is set, the default message is returned.

SetAllTxPathTraceMessages PortHandle PathTraceMessage
Sets the Tx Path Trace message for all the channels on the link. Note that the Path Trace Message may contain escape sequences that will be translated before the message is loaded, typically this includes the channel number and therefore each message will be unique.

IncrementTxPointerPortHandle
Increments the current H1/H2 pointers. If maximum value is reached this wraps around to zero.

DecrementTxPointer ->PortHandle
Decrements the current H1/H2 pointers. If minimum value is reached this wraps around to the maximum value.

SetNewTxPointerValue PortHandle PointerValue NDFState
Sets the current H1/H2 pointers to an arbitrary value.

GetCurrentTxPointerValue PortHandle -> PointerValue

SetCurrentRxChannel PortHandle Sts1Pos
The hardware can only monitor one STS channel at a time. This method switches the receiver circuits to look at a different STS. To ensure that invalid cached information is not returned, the current values of the POH and J1 message are discarded if a channel change occurs.

GetCurrentRxChannel PortHandle -> Sts1Pos
Returns the currently monitored STS channel.

GetRxPathOverheadByte PortHandle Byte -> Sts1Pos Value
Returns the current POH byte value from the current Rx channel. This may return an error if the POH has not been updated following the previous switch of Rx channel. Not available for J1 and B3 bytes.

GetRxPathOverheadHeader PortHandle -> Sts1Pos PathOverheadHeader
Returns the current POH values from the currently selected Rx channel. This may return an error if the POH has not been updated following the previous switch of Rx channel.

SetRxExpectedPathTraceMessageLength PortHandle Sts1Pos TraceLength
Sets the expected receive length of the message. If a call to this function changes the currently set message length the expected receive message will be reset to the default message for the new length.

GetRxExpectedPathTraceMessageLength PortHandle Sts1Pos -> TraceLength

GetRxPathTraceMessage PortHandle -> Sts1Pos TraceLength PathTraceMessage
Returns the received J1 path trace message for the currently selected Rx channel. This may return an error if the J1 Trace message has not been updated following the previous switch of Rx channel.

SetRxExpectedPathTraceMessage PortHandle Sts1Pos PathTraceMessage
At initialization the user defined messages are set to the default message. This method should be used to overwrite that with a user defined message.

GetRxExpectedPathTraceMessage PortHandle Sts1Pos -> PathTraceMessage
Returns the expected J1 trace message for the given STS. After re-configuring a channel or channels to have the expected J1 Trace Message 'as received' there will be a period when this information is unavailable (and error is returned in the case).

SetAllRxExpectedPathTraceMessagesAsReceived PortHandle
All the channels defined in the Rx Channel mask have their expected J1 trace message set to match that currently received. Note that the length of the trace message may be changed by this command, depending on what is received from the link.

SetRxExpectedPathTraceMessageAsReceived PortHandle Sts1Pos
Sets the expected trace message for the given STS to match the currently received trace message.

StartPathTraceMessageCheck PortHandle
Starts a single scan of those channels included in the Receive channel mask STS and validates that the expected message matches that received.

IsPathTraceMessageCheckOn PortHandle -> BOOL
Indicates if there is a Path Routing Message Check in progress.

IsRxPathTraceMessageAsExpected PortHandle Sts1Pos -> Result (BOOL)
Returns the result of the last (which could be a considerable time before) call to StartPathTraceMessageCheck for the requested channel. If the last Path Trace Message Check has not completed, this interface method returns an error

indicating that the check is not complete. Channels not included in the received channel mask (and therefore not checked) also return an error.

`IsMultiRxPathTraceMessageAsExpected PortHandle -> ErroredChannelList`

Returns the result of the last (which could be a considerable time before) call to StartPathTraceMessageCheck for the requested channel. An error is returned if the channel configuration has changed since the last call (as the results could not be relied on). If the last Path Trace Message Check has not completed, an error is returned indicating the fact. Channels not included in the received channel mask do not show any errors.

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector
Byte	Enum	EAgtXmPathOverheadByte • AGT_XM_SONET_J1 • AGT_XM_SONET_B3 • AGT_XM_SONET_C2 • AGT_XM_SONET_G1 • AGT_XM_SONET_F2 • AGT_XM_SONET_H4 • AGT_XM_SONET_Z3 • AGT_XM_SONET_Z4 • AGT_XM_SONET_N1 EAgtXmPathOverHeadByte • AGT_XM_SDH_J1 • AGT_XM_SDH_B3 • AGT_XM_SDH_C2 • AGT_XM_SDH_G1 • AGT_XM_SDH_F2 • AGT_XM_SDH_H4 • AGT_XM_SDH_F3 • AGT_XM_SDH_K3 • AGT_XM_SDH_N1 •
ByteMode	Enum	EAgtXmPath OverheadByteMode • AGT_XM_EDITABLE_BYTE: This byte has a default, but can be user-edited. Available for all the bytes except J1 & B3 bytes. AGT_XM_FIXED_BYTE: This byte is not user editable. Includes J1 & B3 bytes.
Sts1Pos	Long	Limits depend on current Line rate. 10G -- [1-192] 2.5G -- [1-48] 622M -- [1-12] 155M -- [1-3]
Value	unsigned char	8-bit integer (unsigned char) specifying the value of the overhead byte.
PointerValue	Long	[0-782]

NDFState	Bool	1:Flag in On 0:Flag is Off
TraceLength	Enum	EAgtXmTraceLength • AGT_XM_PATH_TRACE_16_BYTES • AGT_XM_PATH_TRACE_64_BYTES
PathTraceMessage	String	User defined string. Could be up to (and including) 15 or 62 bytes long depending on the current selected setting of trace mode. The appropriate terminator will be added. For methods that set the Path Trace Message (Tx and expected Rx) the string may contain escape sequences which will be replaced before the Trace message is used. For the corresponding 'get' methods the expanded string will be returned. These escape sequences can appear anywhere in the message (multiple times if you want). They are all fixed width and the escape sequence reflects the number of characters that the field takes. <inst> - Instrument number (6 characters, taken from configured name). <port> - Port number (6 characters in the format nnnn/n, which is made up of the rack position, module number and the physical port within the module). <c> - Channel number (3 digits, leading 0 added if needed).
PathOverheadHeader	Array	Array of [1 X 9] bytes
ErroredChannelList	List<long>	List of channels with errored J1 bytes.
Result	Bool	1: If Received Path trace string matched with the expected path trace string 0: Otherwise

Notes :

All functions that use parameters with fixed ranges will return an error if called with that parameter out of range. The Sts1Pos parameter's upper range limit is checked dependent on the line rate of the current port.

All methods are valid in both the SONET (AgtXmSonetPathOverhead) and SDH (AgtXmSdhPathOverhead) interfaces. The operation is the same whichever interface is called except where stated. In which case the operation is appropriate to the interface called.

SONET POH Header Default Values

Default values for the SONET POH header (excluding J1 and B3) are:

- C2 - 0x01 - Equipped non specific
- G1 - 0x00 - No B3 errors and no defects
- F2 - 0x00 - No message
- H4 - 0x00 - Not VT-structured format
- Z3 - 0x00 - Reserved for future use
- Z4 - 0x00 - Reserved for future use
- N1 - 0x00 - No incoming error count

For SDH all values are the same except for the C2 byte which is (by default) set to 0xfe - O.181 bulk filled.

The default Path Trace Messages are (both transmit and expected):

16 byte format: Agt <port>-<c>

64 byte format: Agilent OmniBER XM <inst> Port <port>-<c>

Escape sequences will be expanded, padding and appropriate terminators will be added automatically.

The StartPathTraceMessageCheck, SetCurrentRxChannel are two methods that mark the result data as invalid. When switching Rx channel, the current POH and J1 trace messages are marked as invalid until the stored values are updated with new information received from the driver. Similarly the results of any previous path trace message check are marked as invalid when the new check starts. Marking the data as invalid allows a polling user interface to check when the results are ready. Calling the appropriate 'GetData' method after issuing the command may result in the 'Data Not Available' error being returned. If this happens it indicates that the results are not yet ready, and to try the call again. If the user interface has requested callback notification, then the notification routine will be called immediately the results are available.

Error Codes

Calls to these interface methods may return an error. The following error codes are used.

E_AGT_INVALID_PARAMETER - Returned when one of the input parameters is not valid. Typically this will indicate that the channel number is not valid. The number of valid channels depends on the link speed.

E_AGT_RESOURCE_IN_USE - Returned when a second call is made when the first one has yet to complete. For example only one Path Trace Message Check can be in progress at any one time.

E_AGT_DATA_NOT_AVAILABLE - Indicates that the results data from the previous call is not yet available. If polling the interface and try again in a short while, otherwise wait for the callback routine to indicate the data is available.

E_AGT_OUT_OF_BOUNDS - Returned when requesting the multiple results of the last Path Trace Message Check. It indicates that the Rx Channel Mask has changed since the last check and the stored results are no longer valid. Start the Path Trace Message Check again and wait for the results. In the single channel results it indicates that the channel was not included in the last check.

AgtXmSdhPathOverhead

See AgtXmSonetPathOverhead. All the same functions will exist with the same names. The user must use SDH names to reference the bytes in the POH. Where a functional difference occurs between the Sonet and SDH versions, the action taken will be appropriate to the interface used (in this case it overrides the currently selected link level mode).

AgtXmPayload

Summary Configures the payload of ALL the selected transmit channels on a particular port. All the functions are available in Terminal mode but only the Rx functions are available in Thru Mode.

Methods

```
SetTxPayloadType PortHandle PayloadType ChannelPositions[]
GetTxPayloadType PortHandle ChannelPositions[] ->
PayloadTypes[]

SetTxPayloadUserPattern PortHandle PayloadPattern
GetTxPayloadUserPattern PortHandle -> PayloadPattern

SetRxPayloadType PortHandle PayloadType ChannelPositions[]
GetRxPayloadType PortHandle ChannelPositions -> PayloadTypes

SetRxExpectedPayloadUserPattern PortHandle PayloadPattern
GetRxExpectedPayloadUserPattern PortHandle -> PayloadPattern
```

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector.
PayloadType	Enum	AGT_XM_PAYLOAD_PRBS_23 AGT_XM_PAYLOAD_PRBS_23_INVERTED AGT_XM_PAYLOAD_USER_PATTERN AGT_XM_PAYLOAD_USER_LIVE
PayloadPattern	Long	16 bit number
ChannelPosition	Long	List of channel start positions to set to payload type.

AgtXmSonetStatistics

Summary This interface deals with SONET statistics. It is available in both Terminal & Thru Mode.

Each port that is selected for collecting statistics will use the Received Selected channel mask associated with the port. For changing the channel selection, *AgtXmSonetChannelConfig::SelectChannels* and AgtXmSonetVtConfig::SelectVts and other functions in this interface are used.

Syntax

```
AgtInvoke AgtXmSonetStatistics Method InParams -> OutParams
```

Methods

```
SelectStatistics StatisticsHandle Statistics
ListSelectedStatistics StatisticsHandle ->
SelectedStatistics
ListAvailableStatistics -> AvailableStatistics

SelectPorts StatisticsHandle PortHandles
ListSelectedPorts StatisticsHandle -> SelectedPorts

GetAccumulatedValues StatisticsHandle -> SamplingInterval
StatisticsResults
```

Summary Gets SONET-layer statistics from selected test ports during or after a test.

Parameters

StatisticsHandle	Long	The handle that is returned by a call to **AgtStatisticsList** Add AGT_STATISTICS_XM_SONET.
Statistics	list<enum>	List of statistics to be selected.
SelectedStatistics	list<enum>	The statistics you have selected so far.
PortHandles	list<long>	A list of the test ports on which statistics have to be gathered.
SelectedPorts	list<long>	The list of test ports selected.
SamplingInterval	Long	The number of measurement intervals that have elapsed since statistics collection started. Provides a means to order and correlate results, and derive average statistics per interval

SelectionChange	Enum	EAgtStatisticsSelectionChange Valid values are: AGT_STATS_SELECTION_CHANGED AGT_PORT_SELECTION_CHANGED
StatisticsResults	List<double>	Ordered list of statistics by port first and channel thereafter.
AvailableStatistics	List<enum>	The SONET statistics you can get for the selected ports (for details, see **EAgtXmSonetStatistics**): **Analysis Mode Statistics** **GR.253 Section Layer Count** AGT_XM_SONET_GR253_SECTION_SEFS_COUNT AGT_XM_SONET_GR253_SECTION_NEAR_CV_COUNT AGT_XM_SONET_GR253_SECTION_NEAR_ES_COUNT AGT_XM_SONET_GR253_SECTION_NEAR_SES_COUNT **GR.253 Line Layer Counts** AGT_XM_SONET_GR253_LINE_NEAR_CV_COUNT AGT_XM_SONET_GR253_LINE_NEAR_ES_COUNT AGT_XM_SONET_GR253_LINE_NEAR_SES_COUNT AGT_XM_SONET_GR253_LINE_NEAR_UAS_COUNT AGT_XM_SONET_GR253_LINE_FAR_CV_COUNT AGT_XM_SONET_GR253_LINE_FAR_ES_COUNT AGT_XM_SONET_GR253_LINE_FAR_SES_COUNT AGT_XM_SONET_GR253_LINE_FAR_UAS_COUNT AGT_XM_SONET_GR253_LINE_PUAS_COUNT

G828 Section Layer Counts

AGT_XM_SONET_G828_SECTION_NEAR_EB_COUNT
AGT_XM_SONET_G828_SECTION_NEAR_ES_COUNT
AGT_XM_SONET_G828_SECTION_NEAR_SES_COUNT
AGT_XM_SONET_G828_SECTION_NEAR_BBE_COUNT
AGT_XM_SONET_G828_SECTION_NEAR_SEP_COUNT
AGT_XM_SONET_G828_SECTION_NEAR_UAS_COUNT

AGT_XM_SONET_G828_SECTION_NEAR_ES_RATIO
AGT_XM_SONET_G828_SECTION_NEAR_SES_RATIO
AGT_XM_SONET_G828_SECTION_NEAR_BBE_RATIO
AGT_XM_SONET_G828_SECTION_NEAR_SEPI_RATIO

G828 Line Layer Counts

AGT_XM_SONET_G828_LINE_NEAR_EB_COUNT
AGT_XM_SONET_G828_LINE_NEAR_ES_COUNT
AGT_XM_SONET_G828_LINE_NEAR_SES_COUNT
AGT_XM_SONET_G828_LINE_NEAR_BBE_COUNT
AGT_XM_SONET_G828_LINE_NEAR_SEP_COUNT
AGT_XM_SONET_G828_LINE_NEAR_UAS_COUNT

AGT_XM_SONET_G828_LINE_NEAR_ES_RATIO
AGT_XM_SONET_G828_LINE_NEAR_SES_RATIO
AGT_XM_SONET_G828_LINE_NEAR_BBE_RATIO
AGT_XM_SONET_G828_LINE_NEAR_SEPI_RATIO

AGT_XM_SONET_G828_LINE_FAR_EB_COUNT
AGT_XM_SONET_G828_LINE_FAR_ES_COUNT
AGT_XM_SONET_G828_LINE_FAR_SES_COUNT
AGT_XM_SONET_G828_LINE_FAR_BBE_COUNT
AGT_XM_SONET_G828_LINE_FAR_SEP_COUNT
AGT_XM_SONET_G828_LINE_FAR_UAS_COUNT

AGT_XM_SONET_G828_LINE_FAR_ES_RATIO
AGT_XM_SONET_G828_LINE_FAR_SES_RATIO
AGT_XM_SONET_G828_LINE_FAR_BBE_RATIO
AGT_XM_SONET_G828_LINE_FAR_SEPI_RATIO

AGT_XM_SONET_G828_LINE_PUAS_COUNT

AGT_XM_SONET_GR253_PATH_NEAR_CV_COUNT
AGT_XM_SONET_GR253_PATH_NEAR_ES_COUNT
AGT_XM_SONET_GR253_PATH_NEAR_SES_COUNT
AGT_XM_SONET_GR253_PATH_NEAR_UAS_COUNT

AGT_XM_SONET_GR253_PATH_FAR_CV_COUNT
AGT_XM_SONET_GR253_PATH_FAR_ES_COUNT
AGT_XM_SONET_GR253_PATH_FAR_SES_COUNT
AGT_XM_SONET_GR253_PATH_FAR_UAS_COUNT
AGT_XM_SONET_GR253_PATH_PUAS_COUNT

AGT_XM_SONET_G828_PATH_NEAR_EB_COUNT
AGT_XM_SONET_G828_PATH_NEAR_ES_COUNT
AGT_XM_SONET_G828_PATH_NEAR_SES_COUNT
AGT_XM_SONET_G828_PATH_NEAR_BBE_COUNT
AGT_XM_SONET_G828_PATH_NEAR_SEP_COUNT
AGT_XM_SONET_G828_PATH_NEAR_UAS_COUNT

AGT_XM_SONET_G828_PATH_NEAR_ES_RATIO
AGT_XM_SONET_G828_PATH_NEAR_SES_RATIO
AGT_XM_SONET_G828_PATH_NEAR_BBE_RATIO
AGT_XM_SONET_G828_PATH_NEAR_SEPI_RATIO

AGT_XM_SONET_G828_PATH_FAR_EB_COUNT
AGT_XM_SONET_G828_PATH_FAR_ES_COUNT
AGT_XM_SONET_G828_PATH_FAR_SES_COUNT
AGT_XM_SONET_G828_PATH_FAR_BBE_COUNT
AGT_XM_SONET_G828_PATH_FAR_SEP_COUNT
AGT_XM_SONET_G828_PATH_FAR_UAS_COUNT

AGT_XM_SONET_G828_PATH_FAR_ES_RATIO
AGT_XM_SONET_G828_PATH_FAR_SES_RATIO
AGT_XM_SONET_G828_PATH_FAR_BBE_RATIO
AGT_XM_SONET_G828_PATH_FAR_SEPI_RATIO

AGT_XM_SONET_G828_PATH_PUAS_COUNT

	Section/Line Statistics AGT_XM_SONET_B1_ERROR_COUNT AGT_XM_SONET_B1_ERROR_RATIO AGT_XM_SONET_B1_ERROR_SECONDS AGT_XM_SONET_B2_ERROR_COUNT AGT_XM_SONET_B2_ERROR_RATIO AGT_XM_SONET_B2_ERROR_SECONDS AGT_XM_SONET_REIL_ERROR_COUNT AGT_XM_SONET_REIL_ERROR_RATIO AGT_XM_SONET_REIL_ERROR_SECONDS
Enum	AGT_XM_SONET_LOS_ERRORED_SECONDS AGT_XM_SONET_LOF_ERRORED_SECONDS AGT_XM_SONET_SEF_ERRORED_SECONDS AGT_XM_SONET_AISL_ERRORED_SECONDS AGT_XM_SONET_RDIL_ERRORED_SECONDS
	High Order Statistics AGT_XM_SONET_B3_ERROR_COUNT AGT_XM_SONET_B3_ERROR_RATIO AGT_XM_SONET_B3_ERROR_SECONDS
	AGT_XM_SONET_BIT_ERROR_COUNT AGT_XM_SONET_BIT_ERROR_RATIO AGT_XM_SONET_BIT_ERROR_SECONDS
	AGT_XM_SONET_REIP_ERROR_COUNT AGT_XM_SONET_REIP_ERROR_RATIO AGT_XM_SONET_REIP_ERROR_SECONDS
	AGT_XM_SONET_LOPP_ERRORED_SECONDS AGT_XM_SONET_AISP_ERRORED_SECONDS AGT_XM_SONET_RDIP_ERRORED_SECONDS AGT_XM_SONET_UNEQP_ERRORED_SECONDS AGT_XM_SONET_PSL_ERRORED_SECONDS AGT_XM_SONET_PDIP_ERRORED_SECONDS AGT_XM_SONET_POINTER_ACTIVITY_ERRORED_ SECONDS AGT_XM_SONET_POINTER_INC_COUNT AGT_XM_SONET_POINTER_DEC_COUNT
	AGT_XM_SONET_SERVICE_DISRUPTION_LAST_TIME AGT_XM_SONET_SERVICE_DISRUPTION_COUNT AGT_XM_SONET_SERVICE_DISRUPTION_MAX_TIME AGT_XM_SONET_H4LOM_SECONDS

VT/TU Parameters

Available Statistics	List <enum>	The SONET statistics you can get for the selected ports.

Low Order Statistics
AGT_XM_SONET_LO_BIP_ERROR_COUNT
AGT_XM_SONET_LO_BIP_ERRORED_SECONDS
AGT_XM_SONET_LO_BIP_ERROR_RATIO
AGT_XM_SONET_LO_REIV_ERROR_COUNT
AGT_XM_SONET_LO_REIV_ERRORED_SECONDS
AGT_XM_SONET_LO_REIV_ERROR_RATIO
AGT_XM_SONET_LO_BIT_ERROR_COUNT
AGT_XM_SONET_LO_BIT_ERRORED_SECONDS
AGT_XM_SONET_LO_BIT_ERROR_RATIO
AGT_XM_SONET_LO_SERVICE_DISRUPTION_LAST_TIME
AGT_XM_SONET_LO_SERVICE_DISRUPTION_COUNT
AGT_XM_SONET_LO_SERVICE_DISRUPTION_MAX_TIME
AGT_XM_SONET_LO_POINTER_ACTIVITY_ERRORED_SECONDS
AGT_XM_SONET_LO_AISV_ERRORED_SECONDS
AGT_XM_SONET_LO_LOPV_ERRORED_SECONDS
AGT_XM_SONET_LO_RDIV_ERRORED_SECONDS
AGT_XM_SONET_LO_UNEQV_ERRORED_SECONDS
AGT_XM_SONET_LO_RFIV_ERRORED_SECONDS
AGT_XM_SONET_LO_PSL_ERRORED_SECONDS

			Sts1Pos1			Sts1Pos2			Sts1Pos192		
	PortStat1	PortStat2	chstat1	chstat2	chstatN	chstat1	chstat2	chstatN	chstat1	chstat2	chstatN
port 1	1	2	1	2	1	1	2	1	1	2	1
port 2	1	2	0	0	0	1	0	1	0	2	3

order of returned stats

Details

1 Create a new statistics handle for statistics (both port and channel level) using:
   - AgtStatisticsList::Add AGT_STATISTICS_XM_SONET for Sonet
   - AgtStatisticsList::Add AGT_STATISTICS_XM_SDH for Sdh

2 Add selected statistics to the statistics handle using:
   - AgtXmSonetStatistics::SelectStatistics StatisticsHandle List<Values from EAgtXmSonetStatistics>
   - AgtXmSdhStatistics::SelectStatistics StatisticsHandle List<Values from EAgtXmSdhStatistics>
3 Select the ports to collect the statistics using:
   - AgtXmSonetStatistics::SelectPorts
   - AgtXmSdhStatistics::SelectPorts
4 Each port has a default (but modifiable) "Received Selected" channel mask. Change the channel selections using:
   - AgtXmSonetChannelConfig::SelectChannels
   - AgtXmSdhChannelConfig::SelectChannels
5 Use AgtTestController::StartTest to start measuring the selected statistics simultaneously over the selected test ports. You can then poll the system for statistical results, using:
   - AgtXmSonetStatistics::GetAccumulatedValues
   - AgtXmSdhStatistics::GetAccumulatedValues
6 Use AgtXmtStatus to check whether an error occurred in the last sampling interval.

Error Codes

0 Success

1 Invalid parameter:
   - StatisticsHandle: Does not correspond to an item in *AgtStatisticsList*.
   - Statistics: The list of statistics you selected contains a statistic that is not defined.
   - PortHandles: The list of ports you selected contains a port handle that was not returned by *AgtPortSelector*, or one that has since been removed from the current session.

AgtXmSdhStatistics

Summary This interface deals with SDH statistics. It is available in both Terminal and Thru Mode.

Each port that is selected for collecting statistics will use the Received Selected channel mask associated with the port. For changing the channel selection, *AgtXmSdhChannelConfig::SelectChannels* and other functions in this interface are used.

Syntax

```
AgtInvoke AgtXmSdhStatistics Method InParams -> OutParams
```

Methods

```
SelectStatistics StatisticsHandle Statistics
ListSelectedStatistics StatisticsHandle ->
SelectedStatistics
ListAvailableStatistics -> AvailableStatistics

SelectPorts StatisticsHandle PortHandles
ListSelectedPorts StatisticsHandle -> SelectedPorts

GetAccumulatedValues StatisticsHandle -> SamplingInterval
StatisticsResults
```

Summary Gets SDH statistics from selected test ports during or after a test.

Parameters

StatisticsHandle	Long	The handle that was returned when you called **AgtStatisticsList** to add a XM SDH statistics item (*AGT_STATISTICS_XM_SDH*) to the master list. You may also call *AgtStatisticsList* to list existing handles.It is not possible to have port and channel statistics in the same handle.
Statistics	list<enum>	The statistics you want to gather.
SelectedStatistics	list<enum>	The statistics you have selected so far.
PortHandles	list<long>	A list of the test ports on which you want statistics.
SelectedPorts	list<long>	The test ports you have selected so far.
SamplingInterval	Long	The number of measurement intervals that have elapsed since statistics collection started. Provides a means to order and correlate results, and derive average statistics per interval

SelectionChange	Enum	EAgtStatisticsSelectionChange Valid values are: AGT_STATS_SELECTION_CHANGED AGT_PORT_SELECTION_CHANGED
StatisticsResults	list<double>	Ordered by port first and channels thereafter. Port 1. All the statistics which are not valid have -1 as placeholder.
AvailableStatistics	List<enum values>	The SDH statistics you can get for the selected ports (for details, see **EAgtXmSdhStatistics**): **MSOH/RSOH Statistics** AGT_XM_SDH_B1_ERROR_COUNT AGT_XM_SDH_B1_ERROR_RATIO AGT_XM_SDH_B1_ERROR_SECONDS AGT_XM_SDH_B2_ERROR_COUNT AGT_XM_SDH_B2_ERROR_RATIO AGT_XM_SDH_B2_ERROR_SECONDS AGT_XM_SDH_MSREI_ERROR_COUNT AGT_XM_SDH_MSREI_ERROR_RATIO AGT_XM_SDH_MSREI_ERROR_SECONDS AGT_XM_SDH_LOS_ERRORED_SECONDS AGT_XM_SDH_LOF_ERRORED_SECONDS AGT_XM_SDH_OOF_ERRORED_SECONDS AGT_XM_SDH_MSAIS_ERRORED_SECONDS AGT_XM_SDH_MSRDI_ERRORED_SECONDS **Analysis Mode Statistics** **GR.253 Section Layer** AGT_XM_SDH_GR253_REGENERATOR_SEFS_COUNT AGT_XM_SDH_GR253_REGENERATOR_NEAR_CV_COUNT AGT_XM_SDH_GR253_REGENERATOR_NEAR_ES_COUNT AGT_XM_SDH_GR253_REGENERATOR_NEAR_SES_COUNT

GR.253 Line Layer

AGT_XM_SDH_GR253_MULTIPLEX_NEAR_CV_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_NEAR_ES_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_NEAR_SES_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_NEAR_UAS_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_FAR_CV_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_FAR_ES_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_FAR_SES_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_FAR_UAS_COUNT
AGT_XM_SDH_GR253_MULTIPLEX_PUAS_COUNT

G828 Section Layer

AGT_XM_SDH_G828_REGENERATOR_NEAR_EB_COUNT
AGT_XM_SDH_G828_REGENERATOR_NEAR_ES_COUNT
AGT_XM_SDH_G828_REGENERATOR_NEAR_SES_COUNT
AGT_XM_SDH_G828_REGENERATOR_NEAR_BBE_COUNT
AGT_XM_SDH_G828_REGENERATOR_NEAR_SEP_COUNT
AGT_XM_SDH_G828_REGENERATOR_NEAR_UAS_COUNT

AGT_XM_SDH_G828_REGENERATOR_NEAR_ES_RATIO
AGT_XM_SDH_G828_REGENERATOR_NEAR_SES_RATIO
AGT_XM_SDH_G828_REGENERATOR_NEAR_BBE_RATIOAG
T_XM_SDH_G828_REGENERATOR_NEAR_SEPI_RATIO

G828 Line Layer

AGT_XM_SDH_G828_MULTIPLEX_NEAR_EB_COUNT
AGT_XM_SDH_G828_MULTIPLEX_NEAR_ES_COUNT
AGT_XM_SDH_G828_MULTIPLEX_NEAR_SES_COUNT
AGT_XM_SDH_G828_MULTIPLEX_NEAR_BBE_COUNT
AGT_XM_SDH_G828_MULTIPLEX_NEAR_SEP_COUNT
AGT_XM_SDH_G828_MULTIPLEX_NEAR_UAS_COUNT

AGT_XM_SDH_G828_MULTIPLEX_NEAR_ES_RATIO
AGT_XM_SDH_G828_MULTIPLEX_NEAR_SES_RATIO
AGT_XM_SDH_G828_MULTIPLEX_NEAR_BBE_RATIO
AGT_XM_SDH_G828_MULTIPLEX_NEAR_SEPI_RATIO

AGT_XM_SDH_G828_MULTIPLEX_FAR_EB_COUNT
AGT_XM_SDH_G828_MULTIPLEX_FAR_ES_COUNT
AGT_XM_SDH_G828_MULTIPLEX_FAR_SES_COUNT
AGT_XM_SDH_G828_MULTIPLEX_FAR_BBE_COUNT
AGT_XM_SDH_G828_MULTIPLEX_FAR_SEP_COUNT
AGT_XM_SDH_G828_MULTIPLEX_FAR_UAS_COUNT

AGT_XM_SDH_G828_MULTIPLEX_FAR_ES_RATIO
AGT_XM_SDH_G828_MULTIPLEX_FAR_SES_RATIO
AGT_XM_SDH_G828_MULTIPLEX_FAR_BBE_RATIO
AGT_XM_SDH_G828_MULTIPLEX_FAR_SEPI_RATIO

AGT_XM_SDH_G828_MULTIPLEX_PUAS_COUNT

GR253 Path Layer
AGT_XM_SDH_GR253_PATH_NEAR_CV_COUNT
AGT_XM_SDH_GR253_PATH_NEAR_ES_COUNT
AGT_XM_SDH_GR253_PATH_NEAR_SES_COUNT
AGT_XM_SDH_GR253_PATH_NEAR_UAS_COUNT
AGT_XM_SDH_GR253_PATH_FAR_CV_COUNT
AGT_XM_SDH_GR253_PATH_FAR_ES_COUNT
AGT_XM_SDH_GR253_PATH_FAR_SES_COUNT
AGT_XM_SDH_GR253_PATH_FAR_UAS_COUNT
AGT_XM_SDH_GR253_PATH_PUAS_COUNT

		G828 Path Layer AGT_XM_SDH_G828_PATH_NEAR_EB_COUNT AGT_XM_SDH_G828_PATH_NEAR_ES_COUNT AGT_XM_SDH_G828_PATH_NEAR_SES_COUNT AGT_XM_SDH_G828_PATH_NEAR_BBE_COUNT AGT_XM_SDH_G828_PATH_NEAR_SEP_COUNT AGT_XM_SDH_G828_PATH_NEAR_UAS_COUNT AGT_XM_SDH_G828_PATH_NEAR_ES_RATIO AGT_XM_SDH_G828_PATH_NEAR_SES_RATIO AGT_XM_SDH_G828_PATH_NEAR_BBE_RATIO AGT_XM_SDH_G828_PATH_NEAR_SEPI_RATIO AGT_XM_SDH_G828_PATH_FAR_EB_COUNT AGT_XM_SDH_G828_PATH_FAR_ES_COUNT AGT_XM_SDH_G828_PATH_FAR_SES_COUNT AGT_XM_SDH_G828_PATH_FAR_BBE_COUNT AGT_XM_SDH_G828_PATH_FAR_SEP_COUNT AGT_XM_SDH_G828_PATH_FAR_UAS_COUNT AGT_XM_SDH_G828_PATH_FAR_ES_RATIO AGT_XM_SDH_G828_PATH_FAR_SES_RATIO AGT_XM_SDH_G828_PATH_FAR_BBE_RATIO AGT_XM_SDH_G828_PATH_FAR_SEPI_RATIO AGT_XM_SDH_G828_PATH_PUAS_COUNT
AvailableStatisticslist	Enum	**High Order Statistics** AGT_XM_SDH_B3_ERROR_COUNT AGT_XM_SDH_B3_ERROR_RATIO AGT_XM_SDH_B3_ERRORED_SECONDS AGT_XM_SDH_BIT_ERROR_COUNT AGT_XM_SDH_BIT_ERROR_RATIO AGT_XM_SDH_BIT_ERRORED_SECONDS AGT_XM_SDH_HPREI_ERROR_COUNT AGT_XM_SDH_HPREI_ERROR_RATIO AGT_XM_SDH_HPREI_ERRORED_SECONDS

AGT_XM_SDH_AULOP_ERRORED_SECONDS
AGT_XM_SDH_AUAIS_ERRORED_SECONDS
AGT_XM_SDH_HPRDI_ERRORED_SECONDS
AGT_XM_SDH_HPUNEQ_ERRORED_SECONDS
AGT_XM_SDH_POINTER_ACTIVITY_ERRORED_SECONDS
AGT_XM_SDH_PSL_ERRORED_SECONDS
AGT_XM_SDH_PDIP_ERRORED_SECONDS
AGT_XM_SDH_POINTER_INC_COUNT
AGT_XM_SDH_POINTER_DEC_COUNT

AGT_XM_SDH_SERVICE_DISRUPTION_MAX_TIME
AGT_XM_SDH_SERVICE_DISRUPTION_LAST_TIME
AGT_XM_SDH_SERVICE_DISRUPTION_COUNT
AGT_XM_SDH_H4LOM_SECONDS

VT/TU Parameters

The SDH statistics you can get for the selected ports.
Low Order Statistics

AGT_XM_SDH_TU_BIP_ERROR_COUNT
AGT_XM_SDH_TU_BIP_ERRORED_SECONDS
AGT_XM_SDH_TU_BIP_ERROR_RATIO
AGT_XM_SDH_LPREI_ERROR_COUNT
AGT_XM_SDH_LPREI_ERRORED_SECONDS
AGT_XM_SDH_LPREI_ERROR_RATIO
AGT_XM_SDH_LO_BIT_ERROR_COUNT
AGT_XM_SDH_LO_BIT_ERRORED_SECONDS
AGT_XM_SDH_LO_BIT_ERROR_RATIO
AGT_XM_SDH_LO_SERVICE_DISRUPTION_LAST_TIME
AGT_XM_SDH_LO_SERVICE_DISRUPTION_COUNT
AGT_XM_SDH_LO_SERVICE_DISRUPTION_MAX_TIME
AGT_XM_SDH_LO_PTR_ACT_SECONDS
AGT_XM_SDH_TU_AIS_SECONDS
AGT_XM_SDH_TU_LOP_SECONDS
AGT_XM_SDH_LP_RDI_SECONDS
AGT_XM_SDH_LP_UNEQ_SECONDS
AGT_XM_SDH_LP_RFI_SECONDS
AGT_XM_SDH_LO_PSL_SECONDS
AGT_XM_SDH_LO_PDH_AIS_SECONDS
AGT_XM_SDH_LO_PDH_LOF_SECONDS

Details

1. Create a new statistics handle for statistics (both port and channel level) using:
   - AgtStatisticsList::Add AGT_STATISTICS_XM_SONET for Sonet
   - AgtStatisticsList::Add AGT_STATISTICS_XM_SDH for Sdh
2. Add selected statistics to the statistics handle using:
   - AgtXmSonetStatistics::SelectStatistics StatisticsHandle List<Values from EAgtXmSonetStatistics>
   - AgtXmSdhStatistics::SelectStatistics StatisticsHandle List<Values from EAgtXmSdhStatistics>
3. Select the ports to collect the statistics using:
   - AgtXmSonetStatistics::SelectPorts
   - AgtXmSdhStatistics::SelectPorts
4. Each port has a default (but modifiable) "Received Selected" channel mask. Change the channel selections using:
   - AgtXmSonetChannelConfig::SelectChannels
   - AgtXmSdhChannelConfig::SelectChannels
5. Use AgtTestController::StartTest to start measuring the selected statistics simultaneously over the selected test ports. You can then poll the system for statistical results, using:
   - AgtXmSonetStatistics::GetAccumulatedValues
   - AgtXmSdhStatistics::GetAccumulatedValues
6. Use AgtXmtStatus to check whether an error occurred in the last sampling interval.

Error Codes

0 Success

1 Invalid parameter:

- StatisticsHandle: Does not correspond to an item in AgtStatisticsList.
- Statistics: The list of statistics you selected contains a statistic that is not defined.
- PortHandles: The list of ports you selected contains a port handle that was not returned by AgtPortSelector, or one that has since been removed from the current session.

AgtXmSonetChannelConfig

Summary

This interface is used to configure the channel mappings on the Transmit & Receive side for a port. When the port is added, the system assigns a default channel configuration, which is the single largest channel type that can be accomodated in the envelope for the signal rate of the port. After that, the user can change the configuration, using the methods provided in the interface below.

A Channel is denoted by an Enumerated type, like AGT_XM_SONET_CHANNEL_STS24C. The channel position is denoted by the starting STS1 position of the channel.

The user can select and unselect channels, within a configuration. All the unselected channels are transmitted with content of 0. The user can also specify Channel Masks, which are a list of channel start positions. Channel Masks can be provided for Statistics, ErrorAlarm, Transmit, and Receive. By using Channel Masks, the user can control which channels are going to be included in the respective functionality for the masks, without altering the channel configuration. The default Channel Mask is ALL selected.

Syntax

`AgtInvoke AgtXmSonetChannelConfig` *`Method InParams -> OutParams`*

Methods

`SetTxChannelConfiguration` *`PortHandle ChannelList`*

`GetTxChannelConfiguration` *`PortHandle -> ChannelList`*

`SetRxChannelConfiguration` *`PortHandle ChannelList`*

`GetRxChannelConfiguration` *`PortHandle -> ChannelList`*

`GetTxChannelAtPosition` *`PortHandle Sts1Pos -> ChannelType`*

`GetRxChannelAtPosition` *`PortHandle Sts1Pos -> ChannelType`*

`SelectChannel` *`PortHandle ChannelMaskType Sts1Pos`*
Note: Sts1Pos is same as channel Id. i.e. a particular channel in a port can be uniquely identified by its start Sts1Pos(ition).

`UnSelectChannel` *`PortHandle ChannelMaskType Sts1Pos`*

`SelectChannels` *`PortHandle ChannelMaskType Sts1PosList`*

UnSelectChannels *PortHandle ChannelMaskType Sts1PosList*

SelectChannelsInRange *PortHandle ChannelMaskType StartSts1Pos EndSts1Pos*

UnSelectChannelsInRange *PortHandle ChannelMaskType StartSts1Pos EndSts1Pos*

SelectAllChannels *PortHandle ChannelMaskType*

UnSelectAllChannels *PortHandle ChannelMaskType*

IsChannelSelected *PortHandle ChannelMaskType Sts1Pos*

GetSelectedChannels *PortHandle ChannelMaskType -> Sts1PosList*

ChangeTxChannelAtPosition *PortHandle Sts1Pos ChannelType*

ChangeRxChannelAtPosition *PortHandle Sts1Pos ChannelType*

GetTxChannelStartPosition *PortHandle Sts1Pos -> StartSts1Pos*

GetRxChannelStartPosition *PortHandle Sts1Pos -> StartSts1Pos*

GetTxChannelStartPositions *PortHandle -> NumConfiguredChannels ConfiguredChannelsList*
GetRxChannelStartPositions *PortHandle -> NumConfiguredChannels ConfiguredChannelsList*

InvertChannelMask *PortHandle ChannelMaskType*

EnableTxStuffColumnOverWrite *PortHandle*

DisableTxStuffColumnOverWrite *PortHandle*

EnableRxStuffColumnOverWrite *PortHandle*

DisableRxStuffColumnOverWrite *PortHandle*

IsTxStuffColumnOverWriteEnabled *PortHandle*

IsRxStuffColumnOverWriteEnabled *PortHandle*

Note: StuffColumnOverwriteMode in only valid for SONET and NOT for SDH. And in particular it affects only STS-1 channels in the payload.

AutoDiscoverRxSignalStructure *PortHandle*

UndoAutoDiscoverRxSignalStructure *PortHandle*

GetAutoDiscoverRxSignalStatePortHandle -> State

Parameters

ChannelList	List of enums	EAgtXmSonetChannel Size of channels, which will be placed by the system beginning from first STS-1. {3,6,3,12,1,1,1,3,6,24} Implies an STS-3 followed by an STS-6, STS-3 and so on. If the sequence is not legal, the system will return an error.
ChannelType	Enum	EAgtXmSonetChannel AGT_SONET_CHANNEL_STS1 = AGT_SDH_CHANNEL_AU3 AGT_SONET_CHANNEL_STS3C = AGT_SDH_CHANNEL_AU4 AGT_SONET_CHANNEL_STS6C = AGT_SDH_CHANNEL_AU4_2C AGT_SONET_CHANNEL_STS9C = AGT_SDH_CHANNEL_AU4_3C AGT_SONET_CHANNEL_STS12C = AGT_SDH_CHANNEL_AU4_4C AGT_SONET_CHANNEL_STS24C = AGT_SDH_CHANNEL_AU4_8C AGT_SONET_CHANNEL_STS48C = AGT_SDH_CHANNEL_AU4_16C AGT_SONET_CHANNEL_STS192C = AGT_SDH_CHANNEL_AU_464C
Sts1Pos	Long	Can vary from [1-192] for the current system (depending on the line rate being used) [1-192] for 10Gbps [1-48] for 2488 Mbps [1-12] for 622 Mbps [1-3] for 155 Mbps
NewPortHandle	Long	Existing and valid port handle for which the new channel mask would be created.
ChannelMaskType	Enum	EAgtXmChannelMask AGT_XM_CHANNELMASK_ERROR_ALARM AGT_XM_CHANNELMASK_SELECTED_TX AGT_XM_CHANNELMASK_SELECTED_RX
ChannelMaskType	EAgtChannel MaskType	Variable of EAgtChannelMaskType

Sts1Pos	Long	Can vary from [1-192] for the current system (depending on the line rate being used)[1-192] for 10Gbps[1-48] for 2488 Mbps[1-12] for 622 Mbps[1-3] for 155 Mbps
Sts1PosList	list<longs>	{i1, i2, i3, i4, i5, i6} Valid values are from [1-192] for 10GHz.
State	Enum EAgtXmAuto DiscoverSignal State	AGT_XM_NEVER_RUN_BEFORE AGT_XM_RUNNING AGT_XM_COMPLETE_FAIL AGT_XM_COMPLETE_SUCCESS

AgtXmSdhChannelConfig

Summary This interface is the SDH equivalent of the SONET one. The only difference between these currently is in the type of parameters they accept. Refer to the AgtXmSonetChannelConfig Interface for further information.

Syntax

```
AgtInvoke AgtXmSdhChannelConfig Method InParams -> OutParams
```

Methods

```
SetTxChannelConfiguration PortHandle ChannelList

GetTxChannelConfiguration PortHandle -> ChannelList

SetRxChannelConfiguration PortHandle ChannelList

GetRxChannelConfiguration PortHandle -> ChannelList

GetTxChannelAtPosition PortHandle Sts1Pos -> ChannelType

GetRxChannelAtPosition PortHandle Sts1Pos -> ChannelType

SelectChannel PortHandle ChannelMaskType Sts1Pos
Note: Sts1Pos is same as channel Id. i.e. a particular
channel in a port can be uniquely identified by its start
Sts1Pos(ition).

UnSelectChannel PortHandle ChannelMaskType Sts1Pos

SelectChannels PortHandle ChannelMaskType Sts1PosList

UnSelectChannels PortHandle ChannelMaskType Sts1PosList

SelectChannelsInRange PortHandle ChannelMaskType
StartSts1Pos EndSts1Pos

UnSelectChannelsInRange PortHandle ChannelMaskType
StartSts1Pos EndSts1Pos

SelectAllChannels PortHandle ChannelMaskType

UnSelectAllChannels PortHandle ChannelMaskType

IsChannelSelected PortHandle ChannelMaskType Sts1Pos
->IsSelected

GetSelectedChannels PortHandle ChannelMaskType ->
Sts1PosList

ChangeTxChannelAtPosition PortHandle Sts1Pos ChannelType

ChangeRxChannelAtPosition PortHandle Sts1Pos ChannelType

GetTxChannelStartPosition PortHandle Sts1Pos -> StartSts1Pos
```

```
GetRxChannelStartPosition PortHandle Sts1Pos -> StartSts1Pos

GetTxChannelStartPositions PortHandle ->
NumConfiguredChannels ConfiguredChannelsList

GetRxChannelStartPositions PortHandle ->
NumConfiguredChannels ConfiguredChannelsList

InvertChannelMask PortHandle ChannelMaskType

CopyChannelMask SrcPortHandle SrcChannelMaskType
DestPortHandle DestChannelMaskType

AutoDiscoverRxSignalStructure PortHandle

UndoAutoDiscoverRxSignalStructure PortHandle
```

AgtXmBurstControl

Summary This interface exposes the commands to control the timed and pulse modes for the error and alarm generation. The alarm or error selection is controlled by the AgtXmSonetAlarm and AgtXmSonetError interfaces (and their SDH counterparts). This interface allows the parameters for the timed and pulse modes to be set.

Syntax

```
AgtInvoke AgtXmBurstControl Method InParams -> OutParams
```

Methods

```
SetTimedParameters PortHandle OnPeriod OffPeriod RepeatCount
```

```
GetTimedParameters PortHandle -> OnPeriod OffPeriod
RepeatCount
```

Default parameters will be a single burst of 1 second (OnPeriod = 1, OffPeriod = 0, RepeatCount = 1). Error Returns E_AGT_RESOURCE_IN_USE implies that the timer is running; E_AGT_OUT_OF_BOUNDS if any of the input parameters are out of range.

```
StartTimedBurst PortHandle
```

```
StopTimedBurst PortHandle
```

```
IsTimedBurstRunning PortHandle -> BOOL
```

Error returns E_AGT_INVALID_OPERATION if the timer is not in the correct mode to start, i.e. neither alarms nor errors are configured to use it or they are not switched on.

```
SetSinglePulseParameters PortHandle PulseCount
```

```
GetSinglePulseParameters PortHandle -> PulseCount
```

Default value PulseCount = 1. Error Returns E_AGT_OUT_OF_BOUNDS if PulseCount is out of range. (For the future. may return E_AGT_OUT_OF_BOUNDS if set PulsedParameters has set the OffCount or OnCount non-zero.)

```
StartPulsedBurst PortHandle
```

The pulsed burst is active for such a short time that there is no requirement for a stop or status methods. Error returns E_AGT_INVALID_OPERATION if the pulse generator is not in the correct mode to start, i.e. alarms are configured to use it and they are not switched on, or LOS alarm type is selected (which uses a different mechanism, and not available in pulsed mode).

```
SetLosTimedParameters PortHandle LosPeriod
```

```
GetLosTimedParameters PortHandle -> LosPeriod
```

Default LosPeriod = 1.0mSec. Error Returns E_AGT_OUT_OF_BOUNDS if LosPeriod is out of range.

```
StartLosTimedBurst PortHandle
```

The timed burst is active for such a short time that there is no requirement for a stop or status methods. Error returns E_AGT_INVALID_OPERATION if the LOS timer is not in the correct mode to start, i.e. alarms are configured to use it and they are not switched on, and LOS alarm type is selected.

Parameters

OnPeriod	Long	Time in seconds that the error/alarm is generated for. Range 1 - 10000.
OffPeriod	Long	Time in seconds that the error/alarm is not generated. Range 1 - 10000 if RepeatCount > 1. May be 0 if RepeatCount = 1 to give a single timed burst of the error/alarm.
RepeatCount	Long	Number of times the on/off cycle is repeated. Range 1-10000.
PulseCount	Long	Pulse length in frames. Range 1-64. Same as 'InitialCount' for multiple pulse mode.
InitialCount	Long	Initial (or single) pulse length in frames. Range 1-64
OffCount	Long	Off period length in frames. Normal range 1-64, 0 for single pulse mode.
OnCount	Long	On period length in frames. Normal range 1-64, 0 for single pulse mode.
LosPeriod	Long	Time in microseconds that LOS alarm to be generated. Range 0.1 to 110 in steps of 0.1mSec.

AgtOpticalInterface

Summary

This interface Turns on/off optical transmit lasers, selects a transmit/receive mode, and selects the clock source.

Syntax

```
AgtInvoke AgtOpticalInterface Method InParams -> OutParams
```

Methods

```
AllLasersOn
AllLasersOff

LaserOn PortHandle
LaserOff PortHandle
IsLaserOn PortHandle -> LaserState

SetPortMode PortHandle PortMode
GetPortMode PortHandle -> PortMode

SetClockSource PortHandle ClockSource
GetClockSource PortHandle -> ClockSource
GetClockState PortHandle -> ClockState

GetOpticalPowerLevel PortHandle -> PowerLevel
GetOpticalPowerLimits PortHandle -> PowerLimitsList
```

Parameters

PortHandle	Long	A handle to a test port, as returned by AgtPortSelector.
LaserState	Bool	Indicates whether the transmit laser is on: 1-- Laser is On 0-- Laser is Off
PortMode	Enum	• AGT_MODE_NORMAL: The module does Multi-channel, mixed payload generation with some patterns, errors and alarms. • AGT_MODE_LOOPBACK: The test port loops back internally, receiving its own transmitted data (Tx -> Rx). • AGT_MODE_LOOPBACK: is Not supported by Alchemy cards. • AGT_MODE_MONITOR: The module Re-transmits received payload, and has the ability to add some errors/alarms to any/all channels as they pass.

ClockSource	Enum	The clock source for the transmitted signal: • AGT_CLOCK_INTERNAL (default): Synchronised with the clock inside the first test module in the Clock daisy chain. The first module uses an internal 10 MHz (+/- 10 ppm) clock and sends heartbeat pulses every 100 ms to synchronise all test modules on the same Clock line. • AGT_CLOCK_RECOVERED: Synchronised with the clock signal recovered in SONET/SDH frames received from the connected SUT interface. The allowable clock sources depends on the Port Mode: 1 AGT_MODE_NORMAL : Internal, Recovered 2 AGT_MODE_MONITOR: Recovered
ClockState	Bool	Specifies whether the selected clock source is present: 1: Clock is present 0: Clock is not present

AgtXmErrorEventLog

Syntax AgtInvoke AgtXmErrorEventLog Method InParams -> OutParams

Methods

`EnableLogging`

`DisableLogging`

`IsLoggingEnabled -> isEnabled`

`SetLogFile LogFileName`

`GetLogFile -> LogFileName`

Specify the pathname of the log file to be used. The log file will be created when the test is started.

`SelectPorts PortsList`

`ListSelectedPorts -> PortsList`

Select the ports to be logged. Cannot be called while the test is running.

`SelectErrors ErrorsList`

`ListAvailableErrors -> ErrorsList`

`ListSelectedErrors -> ErrorsList`

Parameters

isEnabled	Bool	1: Logging is Enabled 0: Logging is Off
LogFile	String	Log file name
StatisticsSelection	Enum	EAgtStatisticsSelectionChange
PortsList	List of Long	Select the ports to be logged

AgtXmAlarmEventLog

Syntax

`AgtInvoke AgtXmAlarmEventLog Method InParams -> OutParams`

Methods

`EnableLogging`

`DisableLogging`

`IsLoggingEnabled -> isEnabled`

`SetLogFile LogFileName`

`GetLogFile -> LogFileName`

Specify the pathname of the log file to be used. The log file will be created when the test is started.

`SelectPorts PortsList`

`ListSelectedPorts -> PortsList`

Select the ports to be logged. Cannot be called while the test is running.

`SelectAlarms AlarmsList`

`ListAvailableAlarms -> AlarmsList`

`ListSelectedAlarms -> AlarmsList`

Parameters

isEnabled	Bool	1: Logging is Enabled 0: Logging is Off
LogFile	String	Log file name
StatisticsSelection	Enum	EAgtStatisticsSelectionChange
PortsList	List of Long	Select the ports to be logged

AgtStatisticsLog

Summary

This interface Records selected statistics to a comma separated file in real time

Syntax

`AgtInvoke AgtStatisticsLog Method InParams -> OutParams`

Methods

`EnableLogging`

`DisableLogging`

Enable/Disable logging to the statistics log file.

`IsLoggingEnabled -> IsLogEnabled`

`SetLogFile LogFile`

`GetLogFile -> LogFile`

Sets the log file name, the default is statisticsN.csv, where N is the session number.

`SetLoggingInterval Interval`

`GetLoggingInterval -> Interval`

Sets the frequency of logging in seconds, the default is 600 (10 minutes)

`SelectPorts PortHandlesList`

`ListSelectedPorts -> PortHandlesList`

`ListAvailableStatistics -> Statistics`

`ListSelectedStatistics -> Statistics`

`SelectStatistics Statistics`

Parameters

IsLogEnabled	Bool	1: Logging is Enabled 0: Logging is Off
Interval	Long	Logging frequency in seconds

AgtStatisticsList

This interface is used to add a new statistics handle to be used with AgtXmStatistics.

AgtXmSequenceCapture

Summary

Any predefined overhead byte group can be selected for sequence capture. Valid options are K1K2 (in any STS-3c) and K1K2K3 (only for first STS-3c). 256 unique values of the selected channel are displayed along with the number of frames for which the value has occurred. This provides a frame-by-frame log of any changes in the selected overhead byte(s). Capture can be run over multiple ports simultaneously, allowing users to correlate events happening at different nodes in the network.

Syntax

```
AgtInvoke AgtXmSequenceCapture Method InParams -> OutParams
```

Methods

```
SetCaptureByteGroup PortHandle CaptureByteGroup Sts3Num
Sts1Num
```

Sets up the Byte (group) which will be captured. Default Value is AGT_XM_CAPTURE_BYTE_GROUP_K1K2. Only other valid option is AGT_XM_CAPTURE_BYTE_GROUP_K1K2K3.

Following table lists the maximum and minimum valid values of these parameters for various line-rates.

	STS-192c	STS-48c	STS-12c	STS-3c
Sts3Num	[1..64]	[1..16]	[1..4]	[1..1]
	[1..3]	[1..3]	[1..3]	[1..3]

Sts1Num has meaning in the context of any particular STS-3c within the payload. For the byte groups that do not require STS-3 and/or STS-1 numbers, any value you specify will be ignored.

```
GetCaptureByteGroup PortHandle -> CaptureByteGroup Sts3Num
Sts1Num
```

This function returns the currently selected Byte (group) for sequence capturing

Byte Group Name	Sts3	Sts1	No of Bytes	Comments
AGT_XM_CAPTURE_BYTE_GROUP_K1K2	Y	-	2	
AGT_XM_CAPTURE_BYTE_GROUP_K1K2K3	-	-	3	

For any specific byte group, the columns marked with 'Y' are required to be filled up with valid values and the ones with '-' will be ignored.

Sequence capture is triggered by Start of Gating.

Sequence Capture takes a minimum of ~32 msec (256 frames) to a maximum of approx. 34 minutes (65535 * 256 frames) for an OC-192. It automatically stops with the end of Gating, although it could also be manually forced to stop before.

```
GetCaptureState PortHandle -> CaptureState
```

```
GetCapturedSequenceData PortHandle -> NumTransitions
NumBytesInByteGroup CaptureData
```

Parameters

PortHandle	Long	A handle to the test port, as returned by AgtPortSelector.
Sts3Num	Long	Indicates the STS3 Number from which the selected bytes will be captured.Valid range depends on the line speed. 10G = [1-64]; 2G5 = [1-16]; 622M = [1-4]; 155M = [1]
Sts1Num	Long	Indicates which STS1 column within the current STS3 should be used. Valid range 1-3.
CaptureByteGroup	Enum	enum EAgtXmCaptureByteGroup AGT_XM_CAPTURE_BYTE_GROUP_K1K2
OverheadByte	Enum	Generic STS-3 naming.
CaptureState	Enum	EAgtXmCaptureState AGT_XM_CAPTURE_STATE_IDLE AGT_XM_CAPTURE_STATE_TRIGGER_ARMED AGT_XM_CAPTURE_STATE_CAPTURING AGT_XM_CAPTURE_STATE_COMPLETE AGT_XM_CAPTURE_STATE_MAX
NumTransitions	Long	Number of transitions since the start of Sequence Capture
NumBytesInByteGroup	Long	Number of Bytes in the currently selected Byte Group. For instance AGT_XM_CAPTURE_BYTE_GROUP_K1K2 has 2 bytes
CaptureData		Cumulative data since the start of Sequence capture. The data is organised as a single array. If there are 'n' transitions and 'm' entries in the byte group. Then number of elements in the array would be n * [m + 1]. The plus one is because of the count of the frames for which the last value occurred.

AgtXmSonetVtConfig

Summary This interface is used to configure the VT channel structure for the transmitter and receiver.

Syntax

```
AgtInvoke AgtXmSonetVtConfig Method InParams -> OutParams
```

Methods

```
SetTxVtGroupConfig PortHandle Sts1 VtGroup VtStructure
GetTxVtGroupConfig PortHandle Sts1 VtGroup -> VtStructure

SetTxVtConfig PortHandle Sts1 VtStructure[]
GetTxVtConfig PortHandle Sts1 -> VtStructure[]

SetTxAllVtConfig PortHandle VtStructure[]
GetTxAllVtConfig PortHandle -> VtStructure[]

SetRxVtGroupConfig PortHandle Sts1 VtGroup VtStructure
GetRxVtGroupConfig PortHandle Sts1 VtGroup -> VtStructure

SetRxVtConfig PortHandle Sts1 VtStructure[]
GetRxVtConfig PortHandle Sts1 -> VtStructure[]

SetRxAllVtConfig PortHandle VtStructure[]
GetRxAllVtConfig PortHandle -> VtStructure[]

SelectVtGroups PortHandle MaskType Sts1 VtGroup[]
SelectVts PortHandle MaskType VtPositions[]
SelectAllVts PortHandle MaskType
SelectedVts PortHandle -> VtPositions[]

UnselectVtGroups PortHandle MaskType Sts1 VtGroup[]
UnselectVts PortHandle MaskType VtPositions[]
UnselectAllVts PortHandle MaskType
UnselectedVts PortHandle -> VtPositions[]

AutoDiscoverRxSignalStructure PortHandle
UndoAutoDiscoverRxSignalStructure PortHandle
GetAutoDiscoverRxSignalState PortHandle -> State

ResetVtConfig PortHandle
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector.
Sts1	Long	Number between 1 and 48 used to indicate the position within the channel structure.
VtGroup	Long	Number between 1 and 7 used to indicate the Vt group within any given STS-1.
VtStructure	Enum	EAgtXmVtType AGT_XM_NO_VT, Used when STS-N has PDH payload AGT_XM_VT2, AGT_XM_VT1_5
MaskType	Enum	EAgtXmChannelMask AGT_XM_CHANNELMASK_SELECTED_RX, AGT_XM_CHANNELMASK_SELECTED_TX, AGT_XM_CHANNELMASK_ERROR_ALARM
VtPositions	Long	List of VT positions formatted as Sts1, VtNumber, where Sts1 is the start position of the STS channel and VtNumber is the position of the VT in the channel. VtNumber is in the range of 11 through 374 representing positions 1,1 through 3,7,4

Details

Each STS-1 must be configured with appropriate number of VT channels e.g. to configure the first STS-1 to be VT2's use the command.

AgtInvoke AgtXmSonetVtConfig SetTxVtConfig PortHandle 1 AGT_XM_VT2

Each STS-1 may contain 28xAGT_XM_VT1_5, 21xAGT_XM_VT2 or 1xAGT_XM_NO_VT (used for Direct Mapping of PDH structure within STS-1).

AgtXmSdhTuConfig

Summary This interface is used to configure the TU channel structure for the transmitter and receiver.

Syntax

```
AgtInvoke AgtXmSdhTuConfig Method InParams -> OutParams
```

Methods

```
SetTxTuGroupConfig PortHandle Stm0 TuGroup TuStructure
GetTxTuGroupConfig PortHandle Stm0 TuGroup -> TuStructure

SetTxTuConfig PortHandle Stm0 TuGroup TuStructure[]
GetTxTuConfig PortHandle Stm0 TuGroup -> TuStructure[]

SetTxAllTuConfig PortHandle TuStructure[]
GetTxAllTuConfig PortHandle -> TuStructure[]

SetRxTuGroupConfig PortHandle Stm0 TuGroup TuStructure
GetRxTuGroupConfig PortHandle Stm0 TuGroup -> TuStructure

SetRxTuConfig PortHandle Stm0 TuStructure[]
GetRxTuConfig PortHandle Stm0 -> TuStructure[]

SetRxAllTuConfig PortHandle TuStructure[]
GetRxAllTuConfig PortHandle -> TuStructure[]

SelectTuGroups PortHandle MaskType Stm0 TuGroup[]
SelectTus PortHandle MaskType TuPositions[]
SelectAllTus PortHandle MaskType
SelectedTus PortHandle -> TuPositions[]

UnselectTuGroups PortHandle MaskType Stm0 TuGroup[]
UnselectTus PortHandle MaskType TuPositions[]
UnselectAllTus PortHandle MaskType
UnselectedTus PortHandle -> TuPositions[]
AutoDiscoverRxSignalStructure PortHandle
UndoAutoDiscoverRxSignalStructure PortHandle
GetAutoDiscoverRxSignalState PortHandle -> State

ResetTuConfig PortHandle
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector
Stm0	Long	Number between 1 and 48 used to indicate the position within the channel structure.
TuGroup	Long	Number between 1 and 7 uesd to indicate the VT group within any given STM-0
TuStructure	Long	EAgtXmTuType AGT_XM_NO_TU AGT_XM_TU3 AGT_XM_TU12 AGT_XM_TU11
MaskType		EAgtXmChannelMask AGT_XM_CHANNELMASK_SELECTED_RX, AGT_XM_CHANNELMASK_SELECTED_TX, AGT_XM_CHANNELMASK_ERROR_ALARM
TuPositions	Long	List of TU positions formatted as Stm0, TuNumber, where Stm0 is the start position of the STM channel and TuNumber is the position of the TU in the channel. TuNumber is in the range of 11 through 374 representing positions 1,1 through 3,7,4. . 100, 200, 300 are used to represent each of the possible TU3's within an AU-4.

Details

Each AU-3/TUG-3 must be configured with appropriate number of TU channels e.g. to configure the first AU-3 to be TU12's use the command.

AgtInvoke AgtXmSonetVtConfig SetTxVtConfig PortHandle 1 AGT_XM_TU12

Each AU-3 may contain 28xAGT_XM_TU11, 21xAGT_XM_TU12 or 1xAGT_XM_NO_VT (used for Direct Mapping of PDH structure within AU-3). Each TUG-3 may contain 28xAGT_XM_TU11, 21xAGT_XM_TU12, or 1xAGT_XM_TU3.

AgtXmSonetVtPathOverhead

Summary This interface is used to configure the Low Order path overhead.

Syntax

```
AgtInvoke AgtXmSonetVtPathOverhead Method InParams ->
OutParams
```

Methods

```
GetTxPathOverheadByteMode PortHandle Sts1 VtNumber Byte ->
ByteMode
SetTxByteToDefaultValue PortHandle Sts1 VtNumber Byte
SetAllTxBytesToDefaultValue PortHandle Sts1 VtNumber
SetAllChannelsAllTxBytesToDefaultValue PortHandle
SetTxPathOverheadByte PortHandle Sts1 VtNumber VtByte
ByteValue
GetTxPathOverheadByte PortHandle Sts1 VtNumber Byte
SetTxPathOverheadHeader PortHandle Sts1 VtNumber
PathOverheadHeader
GetTxPathOverheadHeader PortHandle Sts1 VtNumber ->
PathOverheadHeader

SetTxPathTraceMessageLength PortHandle Sts1 VtNumber
TraceLength
GetTxPathTraceMessageLength PortHandle Sts1 VtNumber ->
TraceLength
SetTxPathTraceMessage PortHandle Sts1 VtNumber
PathTraceMessage
SetTxPathTraceMessageToDefault PortHandle Sts1 VtNumber
GetTxPathTraceMessage PortHandel Sts1 VtNumber ->
PathTraceMessage
SetAllTxPathTraceMessages PortHandle PathTraceMessage

IncrementTxPointer PortHandle VtType
DecrementTxPointer PortHandle VtType
SetNewTxPointerValue PortHandle VtType PointerValue NDFState
GetCurrentTxPointerValue PortHandle VtType -> PointerValue

SetCurrentRxChannel PortHandle Sts1 VtNumber
GetCurrentRxChannel PortHandle -> Sts1 VtNumber

GetRxPathOverheadByte PortHandle Byte -> Sts1 VtNumber
ByteValue
GetRxPathOverheadHeader PortHandle -> Sts1 VtNumber
PathOverheadHeader

SetRxExpectedPathTraceMessageLength PortHandle Sts1 VtNumber
TraceLength
GetRxExpectedPathTraceMessageLength PortHandle Sts1 VtNumber
-> TraceLength
SetRxExpectedPathTraceMessage PortHandle Sts1 VtNumber
PathTraceMessage
GetRxExpectedPathTraceMessage PortHandle Sts1 VtNumber ->
PathTraceMessage
SetAllRxExpectedPathTraceMessagesAsReceived PortHandle
SetRxExpectedPathTraceMessageAsReceived PortHandle Sts1
VtNumber
```

```
GetRxPathTraceMessage PortHandle -> Sts1 VtNumber
TraceLength PathTraceMessage

StartPathTraceMessageCheck PortHandle
IsPathTraceMessageCheckOn PortHandle -> Result (BOOL)
IsRxPathTraceMessageAsExpected PortHandle -> Result (BOOL)
PathTraceMessage
IsMultiRxPathTraceMessageAsExpected PortHandle ->
MatchingChannelList ErroredChannelList
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector.
Sts1	Long	Number between 1 and 48 used to indicate the position within the channel structure.
VtGroup	Long	Number between 1 and 7 used to indicate the VT group within any given STS-1.
VtType	Enum	EAgtXmVtType AGT_XM_NO_VT, - Used when STS-N has PDH payload AGT_XM_VT2, AGT_XM_VT1_5
MaskType	Enum	EAgtXmChannelMask AGT_XM_CHANNELMASK_SELECTED_RX, AGT_XM_CHANNELMASK_SELECTED_TX, AGT_XM_CHANNELMASK_ERROR_ALARM
VtNumber	Long	11 through 74 representing positions 1,1 through 7,4

Name	Type	Description
VtByte	Enum	EAgtXmPathOverheadByte AGT_XM_SONET_V5 ,AGT_XM_SONET_J2, AGT_XM_SONET_Z6, AGT_XM_SONET_Z7, AGT_XM_SONET_J1, AGT_XM_SONET_B3, AGT_XM_SONET_C2, AGT_XM_SONET_G1, AGT_XM_SONET_F2, AGT_XM_SONET_H4, AGT_XM_SONET_Z3, AGT_XM_SONET_Z4, AGT_XM_SONET_N1
J2Msg	String	Alphanumeric string of up to 15 bytes
ByteValue	Byte	Numeric value in the range of 0 to 255
Matching VtList		Sts1, VtNumber list for the positions for each VT where the J2 message **was** as expected.
ErroredVtList		Sts1, VtNumber list for the positions for each VT where the J2 message was **not** as expected.
Pointer Value	Long	

AgtXmSdhTuLoPathOverhead

Summary This interface is used to configure the TU path overhead.

Syntax

```
AgtInvoke AgtXmSdhTuPathOverhead Method InParams -> OutParams
```

Methods

```
GetTxPathOverheadByteMode PortHandle Stm0 TuNumber Byte -> ByteMode

SetTxByteToDefaultValue PortHandle Stm0 TuNumber Byte
SetAllTxBytesToDefaultValue PortHandle Stm0 TuNumber
SetAllChannelsAllTxBytesToDefaultValue PortHandle
SetTxPathOverheadByte PortHandle Stm0 TuNumber TuByte ByteValue
GetTxPathOverheadByte PortHandle Stm0 TuNumber TuByte
SetTxPathOverheadHeader PortHandle Stm0 TuNumber PathOverheadHeader
GetTxPathOverheadHeader PortHandle Stm0 TuNumber -> PathOverheadHeader

SetTxPathTraceMessageLength PortHandle Stm0 TuNumber TraceLength
GetTxPathTraceMessageLength PortHandle Stm0 TuNumber -> TraceLength
SetTxPathTraceMessage PortHandle Stm0 TuNumber PathTraceMessage
SetTxPathTraceMessageToDefault PortHandle Stm0 TuNumber
GetTxPathTraceMessage PortHandel Stm0 TuNumber -> PathTraceMessage
SetAllTxPathTraceMessages PortHandle PathTraceMessage

IncrementTxPointer PortHandle TuType
DecrementTxPointer PortHandle TuType
SetNewTxPointerValue PortHandle TuType PointerValue NDFState
GetCurrentTxPointerValue PortHandle TuType -> PointerValue

SetCurrentRxChannel PortHandle Stm0 TuNumber
GetCurrentRxChannel PortHandle -> Stm0 TuNumber

GetRxPathOverheadByte PortHandle Byte -> Stm0 TuNumber ByteValue
GetRxPathOverheadHeader PortHandle -> Stm0 TuNumber PathOverheadHeader

SetRxExpectedPathTraceMessageLength PortHandle Stm0 TuNumber TraceLength
GetRxExpectedPathTraceMessageLength PortHandle Stm0 TuNumber -> TraceLength
SetRxExpectedPathTraceMessage PortHandle Stm0 TuNumber PathTraceMessage
GetRxExpectedPathTraceMessage PortHandle Stm0 TuNumber -> PathTraceMessage
SetAllRxExpectedPathTraceMessagesAsReceived PortHandle
```

```
SetRxExpectedPathTraceMessageAsReceived PortHandle Stm0
TuNumber

GetRxPathTraceMessage PortHandle -> Stm0 TuNumber
TraceLength PathTraceMessage

StartPathTraceMessageCheck PortHandle
IsPathTraceMessageCheckOn PortHandle -> Result (BOOL)
IsRxPathTraceMessageAsExpected PortHandle -> Result (BOOL)
PathTraceMessage
IsMultiRxPathTraceMessageAsExpected PortHandle ->
MatchingList ErroredList
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port as returned by AgtPortSelector
Stm0	Long	Number between 1and 48 used to indicate the position within the channel structure.
TuType	Enum	EAgtXmTuType AGT_XM_NO_TU, // Used when AU-3/4 has PDH payload AGT_XM_TU3, AGT_XM_VT2, AGT_XM_VT1_5
MaskType	Enum	EAgtXmChannelMask AGT_XM_CHANNELMASK_SELECTED_RX, AGT_XM_CHANNELMASK_SELECTED_TX, AGT_XM_CHANNELMASK_ERROR_ALARM
TuNumber	Long	11 through 374 representing positions 1,1 through 3,7,4.

Name	Type	Description
TuByte	Enum	EAgtXmPathOverheadByte AGT_XM_SDH_V5, AGT_XM_SDH_J2, AGT_XM_SDH_N2, AGT_XM_SDH_K4, AGT_XM_SDH_J1, AGT_XM_SDH_B3, AGT_XM_SDH_C2, AGT_XM_SDH_G1, AGT_XM_SDH_F2, AGT_XM_SDH_H4, AGT_XM_SDH_F3, AGT_XM_SDH_K3, AGT_XM_SDH_N1
ByteValue	Byte	Numeric value in the range of 0 to 255
MatchingList		Stm0,TuNumber list for the positions for each TU where the J2 message **was** as expected.
ErroredList		Stm0, TuNumber list for the positions for each TU where the J2 message was **not** as expected.
PointerValue	Long	

AgtXmLoPayload

Summary Configures the payload of ALL the selected transmit channels of any given port.

Methods

```
SetTxPayloadType PortHandle PayloadType LoPositions[]
GetTxPayloadType PortHandle LoPositions[] -> PayloadType[]

SetTxPayloadUserPattern PortHandle PayloadPattern
GetTxPayloadUserPattern PortHandle -> PayloadPattern

SetRxPayloadType PortHandle PayloadType LoPositions[]
GetRxPayloadType PortHandle LoPositions[] -> PayloadType[]

SetRxExpectedPayloadUserPattern PortHandle PayloadPattern
GetRxExpectedPayloadUserPattern PortHandle -> PayloadPattern
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector
PayloadType	Enum	EAgtXmPayloadType AGT_XM_PAYLOAD_PRBS_23 AGT_XM_PAYLOAD_PRBS_23_INVERTED AGT_XM_PAYLOAD_USER_PATTERN AGT_XM_PAYLOAD_USER_LIVE
PayloadPattern	Long	16 bit value
LoPositions		List of Vt/Tu start positions to set to payload type.

AgtXmLoSettings

Summary This interface is used to configure the Low Order settings.

Syntax

```
AgtInvoke AgtXmLoSettings Method InParams -> OutParams
```

Methods

```
SetTxMapping PortHandle Type Mapping
GetTxMapping PortHandle Type -> Mapping

SetRxMapping PortHandle Type Mapping
GetRxMapping PortHandle Type -> Mapping
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to aport, as returned by AgtPortSelector
Type	Enum	EAgtXmType AGT_XM_NO_VT, Used when STS-N has PDH payload AGT_XM_VT2, AGT_XM_VT1_5,
Mapping	Enum	EAgtXmMappingType AGT_XM_DS1_ASYNC_SF ,AGT_XM_DS1_ASYNC_ESF, AGT_XM_BULK_FILLED, AGT_XM_E1_ASYNC_UNFRAMED, AGT_XM_E1_ASYNC_CRC_ON, AGT_XM_E1_ASYNC_CRC_OFF, AGT_XM_DS3_ASYNC_UNFRAMED, AGT_XM_DS3_ASYNC_M23, AGT_XM_DS3_ASYNC_CBP, AGT_XM_E3_ASYNC_UNFRAMED

AgtXmTuSettings

Summary This interface is used to configure the TU settings.

Syntax

```
AgtInvoke AgtXmTuSettings Method InParams -> OutParams
```

Methods

```
SetTxMapping  PortHandle Type Mapping
GetTxMapping PortHandle Type -> Mapping

SetRxMapping PortHandle Type Mapping
GetRxMapping PortHandle Type -> Mapping
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector.
Type	Enum	EAgtXmTuType AGT_XM_NO_TU, // Used when AU-N has PDH payload AGT_XM_TU3 AGT_XM_TU12 AGT_XM_TU11
Mapping	Enum	EagtXmMappingType AGT_XM_DS1_ASYNC_SF, AGT_XM_DS1_ASYNC_ESF, AGT_XM_BULK_FILLED, AGT_XM_E1_ASYNC_UNFRAMED, AGT_XM_E1_ASYNC_CRC_ON, AGT_XM_E1_ASYNC_CRC_OFF, AGT_XM_DS3_ASYNC_UNFRAMED, AGT_XM_DS3_ASYNC_M23, AGT_XM_DS3_ASYNC_CBP, AGT_XM_E3_ASYNC_UNFRAMED

AgtXmOptionController

Syntax

```
AgtInvoke AgtXmOptionController Method InParams -> OutParams
```

Methods

```
SetOption PortHandle serial option key
DisableOption PortHandle serial option key
GetOption PortHandle -> Enabled Options
IsOptionEnabled PortHandle option -> State (BOOL)
```

Parameters

Name	Type	Description
PortHandle	Long	A handle to a port, as returned by AgtPortSelector
serial	String	The serial number of the hardware
option	String	The option to be Set / disabled

Supported Datatypes

The functionality of the System layer is accessed by the GUI , TCL and SCPI clients using Microsoft's Component Object Model (COM). Combining this with the use of only Automation data types provides an interface, which can be used by many different languages.

Index

Q

R

S

T

V

Printed in U.K. 03/04

J7241-90012

Agilent Technologies