COM Automation

Online Help
Notices

© Agilent Technologies, Inc. 2001-2009
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.

Manual Part Number
Version 03.82.0000

Edition
April 10, 2009
Available in electronic format only
Agilent Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

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

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

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.
COM Automation—At a Glance

The Agilent Logic Analyzer application includes the COM Automation Server. This software lets you write programs that control the Agilent Logic Analyzer application from remote computers on the Local Area Network (LAN).

- COM Automation Overview (see page 13)
- Setting Up for COM Automation (see page 15)
  - Step 1. Install the LA COM Automation client software (see page 34)
  - Step 2. Test your Distributed COM connection (see page 35)
  - Distributed COM Troubleshooting (see page 36)
- Using COM Automation (see page 41)
  - Using Visual Basic for Applications (VBA) in Microsoft Excel (see page 42)
  - Using Visual Basic (in Visual Studio) (see page 45)
  - Example Visual Basic and Visual C++ Programs (see page 46)
  - Using Visual C++ (see page 70)
  - Using LabVIEW (see page 72)
  - Using Perl (see page 76)
  - Using Python (see page 80)
  - Using Tcl (see page 84)
- Reference (see page 89)
  - Objects, Methods, and Properties Quick Reference (see page 90)
  - Object Hierarchy Overview (see page 102)
  - Objects (Quick Reference) (see page 105)
  - Methods (see page 166)
  - Properties (see page 243)
- What’s Changed (see page 293)
Contents

COM Automation—At a Glance 3

1 COM Automation Overview

2 Setting Up for COM Automation

   Changing Firewall Settings 16
     Changing Windows Firewall Settings, XP Service Pack 2 16
     Changing Firewall Settings, XP Service Pack 1 20
     Changing Firewall Settings, Vista Service Pack 1 25

   Supported Networking Configurations 33

   Step 1. Install the LA COM Automation client software 34
   Step 2. Test your Distributed COM connection 35

   Distributed COM Troubleshooting 36
     To turn off simple file sharing 37
     To verify logic analyzer machine-wide Distributed COM properties 37
     To verify logic analyzer application Distributed COM properties 38
     To verify remote computer application Distributed COM properties 39

3 Using COM Automation

   Using Visual Basic for Applications (VBA) in Microsoft Excel 42
   Using Visual Basic (in Visual Studio) 45
   Example Visual Basic and Visual C++ Programs 46
     Loading, Running, Storing 46
     Setting Up Simple Triggers 51
     Setting Up Advanced Triggers 54
     Changing the Sampling Mode 58
     Checking the Logic Analyzer Software Version 62
     Additional Visual Basic Examples 69

   Using Visual C++ 70

   Using LabVIEW 72
     Tutorial - To programmatically control the logic analyzer in LabVIEW 72
     LabVIEW Examples 75

   Using Perl 76
4 COM Automation Reference

Objects, Methods, and Properties Quick Reference  90
Object Hierarchy Overview  102
Object Quick Reference  105
   AnalyzerModule Object  106
   BusSignal Object  108
   BusSignalData Object  108
   BusSignalDifference Object  109
   BusSignalDifferences Object  109
   BusSignals Object  113
   CompareWindow Object  118
   Connect Object  118
   ConnectSystem Object  119
   FindResult Object  119
   Frame Object  120
   Frames Object  120
   Instrument Object  121
   Marker Object  123
   Markers Object  123
   Module Object  128
   Modules Object  129
   PattgenModule Object  129
   Probe Object  135
   Probes Object  135
   SampleBusSignalData Object  138
   SampleDifference Object  149
   SampleDifferences Object  149
   SelfTest Object  149
   Tool Object  152
   Tools Object  153
   VbaViewChart Object  156
   VbaViewChartAxis Object  157
   VbaViewChartData Object  157
   VbaViewChartFont Object  158
   VbaViewChartLegend Object  158
   VbaViewChartTitle Object  159
   VbaViewWebBrowser Object  159
   VbaViewWindow Object  160
   Window Object  160
   Windows Object  161
Methods 166
Add Method (BusSignals Object) 168
Add Method (Markers Object) 169
AddXML Method 170
AddPointArrays Method 170
Clear Method (for VbaViewChartData object) 171
Clear Method (for VbaViewWebBrowser object) 171
ClearOutput Method 172
Close Method 172
Connect Method 172
CopyFile Method 173
DeleteFile Method 173
DoAction Method 174
DoCommands Method 174
Draw Method 178
Execute Method 178
Export Method 179
ExportEx Method 180
Find Method 184
FindNext Method 188
FindPrev Method 189
GetDataBySample Method 189
GetDataByTime Method 192
GetGroupCaption Method 193
GetLine Method 194
GetLineLabel Method 196
GetModuleByName Method 196
GetNumSamples Method 199
GetProbeByName Method 200
GetRawData Method 200
GetRawTimingZoomData Method 202
GetRemoteInfo Method 203
GetSampleNumByTime Method 204
GetTime Method 204
GetToolByName Method 205
GetValueCaption Method 206
GetWindowByName Method 206
GoOffline Method 207
GoOnline Method 211
GoToPosition Method 212
Import Method 212
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImportEx Method</td>
<td>213</td>
</tr>
<tr>
<td>InsertLine Method</td>
<td>214</td>
</tr>
<tr>
<td>IsOnline Method</td>
<td>214</td>
</tr>
<tr>
<td>IsTimingZoom Method</td>
<td>215</td>
</tr>
<tr>
<td>New Method</td>
<td>215</td>
</tr>
<tr>
<td>Open Method</td>
<td>216</td>
</tr>
<tr>
<td>PanelLock Method</td>
<td>216</td>
</tr>
<tr>
<td>PanelUnlock Method</td>
<td>220</td>
</tr>
<tr>
<td>QueryCommand Method</td>
<td>220</td>
</tr>
<tr>
<td>RecallTriggerByFile Method</td>
<td>223</td>
</tr>
<tr>
<td>RecallTriggerByName Method</td>
<td>224</td>
</tr>
<tr>
<td>RecvFile Method</td>
<td>224</td>
</tr>
<tr>
<td>Remove Method (BusSignals Object)</td>
<td>224</td>
</tr>
<tr>
<td>Remove Method (Markers Object)</td>
<td>225</td>
</tr>
<tr>
<td>RemoveAll Method</td>
<td>225</td>
</tr>
<tr>
<td>RemoveXML Method</td>
<td>225</td>
</tr>
<tr>
<td>RemoveLine Method</td>
<td>226</td>
</tr>
<tr>
<td>Reset Method (PattgenModule Object)</td>
<td>226</td>
</tr>
<tr>
<td>Resume Method (PattgenModule Object)</td>
<td>227</td>
</tr>
<tr>
<td>Run Method (Instrument Object)</td>
<td>227</td>
</tr>
<tr>
<td>Run Method (PattgenModule Object)</td>
<td>228</td>
</tr>
<tr>
<td>Save Method</td>
<td>228</td>
</tr>
<tr>
<td>SendFile Method</td>
<td>229</td>
</tr>
<tr>
<td>SetGroupCaption Method</td>
<td>229</td>
</tr>
<tr>
<td>SetLine Method</td>
<td>230</td>
</tr>
<tr>
<td>SetLineLabel Method</td>
<td>230</td>
</tr>
<tr>
<td>SetValue Method</td>
<td>231</td>
</tr>
<tr>
<td>SetValueArray Method</td>
<td>231</td>
</tr>
<tr>
<td>SetValueCaption Method</td>
<td>232</td>
</tr>
<tr>
<td>SimpleTrigger Method</td>
<td>232</td>
</tr>
<tr>
<td>Step Method</td>
<td>235</td>
</tr>
<tr>
<td>Stop Method (Instrument Object)</td>
<td>236</td>
</tr>
<tr>
<td>Stop Method (PattgenModule Object)</td>
<td>236</td>
</tr>
<tr>
<td>TestAll Method</td>
<td>237</td>
</tr>
<tr>
<td>VBADisplayHelpTopic Method</td>
<td>237</td>
</tr>
<tr>
<td>VBARunMacro Method</td>
<td>237</td>
</tr>
<tr>
<td>VBARunRPICommand Method</td>
<td>238</td>
</tr>
<tr>
<td>WaitComplete Method</td>
<td>238</td>
</tr>
<tr>
<td>WriteOutput Method</td>
<td>242</td>
</tr>
</tbody>
</table>
In some test and measurement environments, the process of making a measurement and analyzing the results become routine or repetitive. In other environments, it may be more convenient to make measurements or analyze data from a remote PC. Whatever the situation, you can benefit from performing measurement tasks programmatically through a Visual C++ or Visual Basic program.

The COM Automation Server is part of the Agilent Logic Analyzer application. It gives PC applications a COM interface to the logic analyzer (see Figure 1). This lets you write programs that communicate with the logic analyzer using a COM model definition and take advantage of the ease of programming offered by the Visual Studio Environment (that is, Visual Basic or Visual C++).

By executing programs using the Instrument COM Automation Server, you manipulate the logic analysis environment and its functional components as Objects. You manipulate objects by using the properties and methods associated with the objects. Methods represent actions you take against the objects. Properties represent characteristics of the objects, such as their type or size.

Each object implements a dual interface through which you can manipulate the object. Each object implements an IDispatch interface for Automation and a Component Object Model (COM) interface for direct access to object members (properties and methods). By importing the Instrument Automation Server's type library, you can employ early binding by using the COM interface. Early binding makes all calls into interface members faster at run time.
For more information on Logic Analyzer Objects and the components that manipulate them, refer to Object Hierarchy Overview (see page 102).
A remote computer connected to the logic analyzer via LAN uses Distributed COM (DCOM) to control the logic analyzer. COM is used when the logic analyzer is controlled from within the logic analyzer itself. Because COM connections work without any additional configuration, this getting started section only pertains to Distributed COM.

The following assumptions are made in this getting started section. Verify the validity of each before proceeding.

- You have changed the Windows XP firewall settings (see page 16) on the logic analyzer to allow remote access to the services required for DCOM.
- Both the remote computer and logic analyzer are on the same LAN and both can "see" each other in "My Network Places" (or communicate with each other using the ping command in a Command Prompt window). See supported networking configurations (see page 33) for more information.
- The remote computer should be running Windows XP or Windows Vista.
- The logic analyzer is running version 02.00 or later of the Agilent Logic Analyzer application.
- You are reasonably familiar with Windows.

Setting up your remote computer to communicate with the logic analyzer requires the following three summarized steps:

1. Install the LA COM Automation client software (see page 34) on your remote computer.
2. Test your Distributed COM connection (see page 35).

See Also

- Using COM Automation (see page 41)
Changing Firewall Settings

Depending on when your logic analysis system was shipped from the factory (and whether you have installed the latest operating system updates), you could have Service Pack 1 or Service Pack 2 of the Windows XP Professional operating system installed. Changing firewall settings is different in these two service packs:

- "Changing Windows Firewall Settings, XP Service Pack 2" on page 16
- "Changing Firewall Settings, XP Service Pack 1" on page 20

You can also run the Agilent Logic Analyzer application on a PC with the Windows Vista business or enterprise operating system (as in the performance multiframe configuration). In this case, you can change the firewall settings using the Windows Firewall with Advanced Security control panel applet:

- "Changing Firewall Settings, Vista Service Pack 1" on page 25

Changing Windows Firewall Settings, XP Service Pack 2

At the A.03.00 release of the Agilent Logic Analyzer application, 1680-series logic analyzers and 16900-series logic analysis systems are shipped from the factory with Windows XP Professional, Service Pack 2 (SP2), and the Windows Firewall is enabled and set up with the exceptions required by the logic analysis system.

- To give other applications/ports access through Windows Firewall (see page 16)
- To restore logic analysis system Windows Firewall defaults (see page 18)

To give other applications/ports access through Windows Firewall

For example, you may need to change firewall settings in order to:

- Use NetOp to remotely control the logic analysis system.
- Use RealVNC to remotely control the logic analysis system.

To change firewall settings to give other applications/ports access:

1. From the Windows Start menu, choose Start>Control Panel.
2. In the Control Panel window, open Windows Firewall.
3 In the Windows Firewall dialog, click the **Exceptions** tab.

4 In the Exceptions tab, if the program or service is listed, check its box to enable it; otherwise, click **Add Program...** to give unlisted applications permission to penetrate the firewall, or click **Add Port...** to give unlisted ports access through the firewall.
Refer to the application’s documentation for information on port numbers or other firewall setup information.

**NOTE**

Note that there are separate Windows Firewall profiles: Domain for when the computer has domain membership, and Standard for when the computer has workgroup membership. If you change the type of membership, any changes you made to the Windows Firewall settings will have to be made again.

**See Also**

- To restore logic analysis system Windows Firewall defaults (see page 18)

**To restore logic analysis system Windows Firewall defaults**

1. From the Windows Start menu, choose *Start>*Control Panel.
2. In the Control Panel window, open *Windows Firewall*.
3. In the Windows Firewall dialog, click the *Advanced* tab.
In the Advanced tab, click **Restore Defaults** to restore the default Windows Firewall settings.

In the confirmation dialog, Click **Yes**.

Click **OK** to close the Windows Firewall dialog.

From the Windows Start menu, choose **Start>Run**; then, enter or select the file "C:\Program Files\Agilent Technologies\Logic Analyzer\agFirewSP2.wsf", and click **OK**.

The logic analysis system firewall defaults enable the following exceptions:

- Agilent Logic Analysis Application program.
- Agilent Logic Analysis Service program.
- File and Printer Sharing.
- Microsoft RPC Endpoint Mapper TCP port.
- Remote Desktop service.
- Web Server (HTTP) port.

The logic analysis system firewall defaults also enable the following ICMP (Internet Control Message Protocol) settings:

- Allow incoming echo request.
- Allow outgoing destination unreachable.
- Allow outgoing time exceeded.
Note that there are separate Windows Firewall profiles: Domain for when the computer has domain membership, and Standard for when the computer has workgroup membership. When you restore the logic analysis system firewall defaults, the defaults for both profiles are restored.

See Also • To give other applications/ports access through Windows Firewall (see page 16)

Changing Firewall Settings, XP Service Pack 1

During the A.02.xx releases of the Agilent Logic Analyzer application, 1680-series logic analyzers and 16900-series logic analysis systems were shipped from the factory with Windows XP Professional, Service Pack 1 (SP1), and the Internet Connection Firewall was enabled and set to block accesses to all network services. If you have not yet installed Service Pack 2, you can follow these instructions to change firewall settings.

You need to change firewall settings and allow access to network services in order to:
• Connect to the logic analyzer from the Agilent Logic Analyzer application running on another computer.
• Connect 16900A, 16901A, 16902A, or 16902B logic analysis systems together in a multiframe configuration.
• Control the logic analyzer remotely using COM automation programs.
• Use Remote Desktop to remotely control the logic analysis system.
• Use NetOp to remotely control the logic analysis system.
• Use RealVNC to remotely control the logic analysis system.
• Access shared folders on the logic analysis system.
• Access the logic analysis system's web server.

To change the Windows XP firewall settings:
1 From the Windows Start menu, choose Start>Control Panel.
2 In the Control Panel window, open Network Connections.
3 In the Network Connections window, right-click on the Local Area Connection and choose Properties.
4 In the Advanced tab of the Local Area Connection Properties dialog, click **Settings**.…

5 In the Services tab of the Advanced Settings dialog, check the service that you want to allow access to.
6 Click Edit....
7 In the Service Settings dialog, enter "localhost" in the Name or IP address field, and click OK.

![Service Settings dialog]

The "localhost" will automatically be changed to the correct hostname of your logic analysis system.

8 Repeat steps 5 through 7 for all the services you want to allow access to. The services required for various features are:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Service(s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Connect</td>
<td>• 16900</td>
<td></td>
</tr>
<tr>
<td>Multiframe</td>
<td>• 16900</td>
<td></td>
</tr>
</tbody>
</table>
If you want to connect to the logic analyzer from the Agilent Logic Analyzer application running on another computer, or if you want to use 16900A, 16901A, 16902A, or 16902B logic analysis systems in a multiframe configuration, click the ICMP (Internet Control Message Protocol) tab and select:

- Allow incoming echo request
- Allow outgoing destination unreachable
- Allow outgoing time exceeded
10 Click **OK** to close the Advanced Settings dialog.

11 Click **OK** to close the Local Area Connection Properties dialog.
Changing Firewall Settings, Vista Service Pack 1

If you are running the Agilent Logic Analyzer application on a PC with the Windows Vista business or enterprise operating system (as in the performance multiframe configuration), you can change the firewall to allow Distributed COM by following these steps:

1. Open the Windows Firewall with Advanced Security control panel applet (under Control Panel>System and Maintenance>Administrative Tools>).

2. In the User Account Control dialog, click Continue.
3 In the Windows Firewall with Advanced Security window, select **Inbound Rules**.

4 In the Actions pane to the right of the Inbound Rules list, select **New Rule**...
5 In the New Inbound Rules Wizard:
   a In the Rule Type page, select **Program**; then, click **Next >**.
   
   ![Screenshot of New Inbound Rule Wizard](image)

   b In the Program page, click **Browse** to select the "agLogic.exe" program (the default installation directory is C:\Program Files\Agilent Technologies\Logic Analyzer\); then, click **Next >**.
c In the Action page, select *Allow the connection*; then, click **Next >**.
In the Profile page, select **Domain**, **Private**, and **Public**; then, click **Next >**.
Setting Up for COM Automation

- In the Name page, enter the Name "Agilent Logic Analyzer"; then, click Finish.
6 Add another inbound rule for the "agLogicSvc.exe" program (in the same directory), using the same settings, and give it the name "Agilent Logic Analyzer Service".

When you are done, the Inbound Rules list looks like this:

With the added rules' columns showing these settings:
- Profile: Any
- Enabled: Yes
- Action: Allow
- Override: No
- Local Address: Any
- Remote Address: Any
- Protocol: Any
- Local Port: Any
Remote Port: Any
Allowed Users: Any
Allowed Computers: Any

These inbound rules allow any remote computer to communicate with TCP ports 16900 and 16901 exposed by the agLogicSvc.exe process.

7 Also in the Inbound Rules list, find all instances of the existing rule named "File and Printer Sharing (Echo Request - ICMPv4-In)" (there may be a rule for each firewall profile), and make sure these rules are enabled if they are not already.

The rule should be configured to allow access.

This tells the firewall to allow ICMPv4 incoming echo requests. This allows the computer to be pinged on the network — as well as detected by the Agilent Logic Analyzer application.

8 (No modifications to the Outbound Rules are required.)

9 Close the Windows Firewall with Advanced Security window.
Supported Networking Configurations

There are many ways of setting up your remote computer to communicate with the logic analyzer, but due to the security requirements of Distributed COM, only two configurations are supported. The first configuration is when both the remote computer and the logic analyzer are members of a "Workgroup", and the second configuration is when both the remote computer and the logic analyzer are members of a "Domain".

Agilent 1680-series logic analyzers and 16900-series logic analysis systems are shipped from the factory such that "Everyone" has permission to launch and access the Agilent Logic Analyzer application via Distributed COM. The term "Everyone" refers to a different range of users depending on whether the logic analyzer is a member of a Domain or Workgroup. By default, the logic analyzer is configured as a member of a workgroup. Therefore, "Everyone" includes only those users who have been given logon accounts on the logic analyzer.

**Workgroup**

A workgroup is established by the logic analyzer administrator declaring the workgroup name and declaring the logic analyzer as a member of the workgroup. A workgroup does not require a network administrator to create it or control membership.

"Everyone" includes only those users who have been given logon accounts on the logic analyzer. By default, the logic analyzer is configured as members of a workgroup named WORKGROUP.

**NOTE**

To set up a logon account for a new user, see the operating system’s online help. For Distributed COM access, the user’s account name and password must EXACTLY match their remote computer logon account name and password.

**NOTE**

Recent Windows security patches require passwords to be set on accounts before Distributed COM access is allowed.

**Domain**

A domain is typically a large organizational group of computers. Network administrators maintain the domain and control which machines have membership in it.

"Everyone" includes those people who have membership in the domain. In addition, those with logon accounts can also access the analyzer.

**See Also**

- "Agilent Logic Analyzers Isolated Network Setup White Paper"
Step 1. Install the LA COM Automation client software

Because the logic analyzer and remote computer are on the same LAN, you can install the COM automation client software from the logic analyzer:

1. If you have changed the Windows XP firewall settings (see page 16) on the logic analyzer to allow remote access to shared folders, then on your remote computer, map a network drive to the logic analyzer (for example, \computer-name\C$).

2. Navigate to the \Program Files\Agilent Technologies\Logic Analyzer directory on the mapped network drive.

3. Locate the "SetupLACOM.exe" file, and run it to install the LA COM Automation client software on your remote computer.

Or, you can install the COM automation client software from the Agilent Logic Analyzer install CD:

1. Place the Agilent Logic Analyzer install CD in your CD-ROM drive.

   (If your CD-ROM drive is not set up to auto-run, run the Setup.exe program on the CD.)

2. In the Main Menu page, click Install Products.

3. In the Install Products page, click Install LA COM Automation.

4. Follow the installation program's instructions.

Next:  • Step 2. Test your Distributed COM connection (see page 35)
Step 2. Test your Distributed COM connection

A test program called "COM Connection Tool" was created to test your COM/Distributed COM connection to the logic analyzer. To run the "COM Connection Tool" program:

1. Select Start>All Programs>Agilent Logic Analyzer>Utilities>COM Automation>COM Connection Tool.

2. Enter the hostname or IP address of the logic analyzer you want to connect to; then, press the Connect button.

3. If there is a problem with the COM connection, check the status information and click More Info for additional information about the possible causes of the error.

If the COM Connection Tool does not resolve the problem, see also Distributed COM Troubleshooting (see page 36).
Distributed COM Troubleshooting

If the Client Test program (see page 35) fails to connect to the logic analyzer:

- Make sure the remote computer and logic analyzer can "see" each other in "My Network Places" (or communicate with each other using the `ping` command in a Command Prompt window).
- Make sure the logic analyzer is running version 02.00 or later of the Agilent Logic Analyzer application.
- See supported networking configurations (see page 33). If you are in a Workgroup, check that both the account name and password used on both the logic analyzer and remote computer match EXACTLY. Also, if you are in a Workgroup and have the Windows XP operating system, you must turn off simple file sharing (see page 37).

**NOTE**
Recent Windows security patches require passwords to be set on accounts before Distributed COM access is allowed.

- Make sure you are logged in to the logic analyzer so the user and privileges are assigned correctly. The Agilent Logic Analyzer application does not have to be running; if it isn't, connecting via COM will automatically start it.
- To verify logic analyzer machine-wide Distributed COM properties (see page 37)
- To verify logic analyzer application Distributed COM properties (see page 38)
- To verify remote computer application Distributed COM properties (see page 39)
- Make sure the logic analyzer allows DCOM access though the firewall (see Changing Firewall Settings (see page 16)). Some IT departments are now automatically installing firewalls onto remote client computers. Verify your remote client computer also allows DCOM access through the firewall if one is installed.
- If you have a remote client computer that cannot connect to the logic analyzer and one that can, run `ipconfig -all` in the Command Prompt window on both computers to see how their LAN configurations differ. This may help in troubleshooting the problem.
To turn off simple file sharing

If both the remote computer and the logic analyzer are members of a "Workgroup" (see supported networking configurations (see page 33)) and the logic analyzer has the Windows XP operating system, you must turn off simple file sharing.

1. Open Windows Explorer (or double-click My Computer).
2. From the Windows Explorer menu, choose Tools>Folder Options....
3. In the Folder Options dialog, select the View tab.
4. In the "Advanced settings" options list, uncheck Use simple file sharing (Recommended).
5. Click OK to close the Folder Options dialog.

To verify logic analyzer machine-wide Distributed COM properties

Normally, the logic analyzer Distributed COM configuration is set at the factory. If this has been changed, you may have to set it back to the default settings.

To verify the machine-wide Distributed COM properties on the computer that runs the Agilent Logic Analyzer application:

1. From the Windows task bar choose Start>Run..., enter DCOMCNFG.EXE as the name of the program to open, and click OK.
2. Access the machine-wide Distributed COM properties, security, and protocols tabs:
   - In the left-side pane of the Component Services window, browse to the Console Root, Component Services, Computers folder; then, right-click on My Computer and choose Properties from the popup menu.
3. In the Default Properties tab:
   - Check the Enable Distributed COM on this computer option.
   - For the Default Authentication Level, select Connect.
   - For the Default Impersonation Level, select Identify.
4. In the COM Security tab:
   - Under Access Permissions, click Edit Limits....
   - In the Access Permission dialog, make sure the Everyone account has "Allow" checked for both Local Access and Remote Access.
   - Click OK to close the Access Permission dialog.
   - Under Launch and Activation Permissions, click Edit Limits....
   - In the Launch Permission dialog, make sure the MACHINE\Administrators and Everyone accounts have "Allow" checked for:
Local Launch, Remote Launch, Local Activation, and Remote Activation.

5 Click OK to close the Launch Permission dialog.

6 In the Default Protocols tab:
   a Make sure Connection-oriented TCP/IP is listed first.
   6 Click OK to close the properties dialog.

To verify logic analyzer application Distributed COM properties

Normally, the logic analyzer Distributed COM configuration is set at the factory. If this has been changed, you may have to set it back to the default settings.

To verify the application's Distributed COM properties on the computer that runs the Agilent Logic Analyzer application:

1 From the Windows task bar choose Start>Run..., enter DCOMCNFG.EXE as the name of the program to open, and click OK.

2 Open the Agilent 168x/169x/169xx Logic Analyzer Properties dialog:
   • In the left-side pane of the Component Services window, browse to Console Root, Component Services, Computers, My Computer, DCOM Config, Agilent 168x/169x/169xx Logic Analyzer; then, right-click and choose Properties from the popup menu.

3 In the Agilent 168x/169x/169xx Logic Analyzer Properties dialog, verify the following settings under each Tab heading indicated below.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Authentication Level should be set to &quot;Default&quot;.</td>
</tr>
<tr>
<td>Location</td>
<td>Set to &quot;Run application on this computer&quot;.</td>
</tr>
</tbody>
</table>
| Security    | Use custom launch and activation permissions. Verify "Everyone" has launch and activation permissions. If not:  
   • Add "Everyone" and make sure "Allow" is checked for Local Launch, Remote Launch, Local Activation, and Remote Activation.  
   Use custom access permissions. Verify "Everyone" has access permission. If not:  
   • Add "Everyone" and make sure "Allow" is checked for Local Access and Remote Access.  
   Use default configuration permissions. |
| Endpoints   | Leave at default system protocols.                                      |
| Identity    | Set to "The interactive user".                                          |
4 In the Agilent 168x/169x/169xx Logic Analyzer Properties dialog, click OK.

**To verify remote computer application Distributed COM properties**

Normally, the remote computer Distributed COM configuration is set when you install the LA COM Automation client software. If this has been changed, you may have to set it back to the default settings.

To verify the application's Distributed COM properties on the remote computer:

1. From the Windows task bar choose **Start>Run...**, enter DCOMCNFG.EXE as the name of the program to open, and click **OK**.
2. Open the Properties dialog for **Agilent 168x/169x/169xx Logic Analyzer**:
   a. In the Component Services window, navigate the hierarchy tree to **Component Services>Computers>My Computer>DCOM Config**.
   b. In the DCOM Config folder, right-click on **Agilent 168x/169x/169xx Logic Analyzer** and choose **Properties** from the popup menu.
3. In the Agilent 168x/169x/169xx Logic Analyzer Properties dialog, verify the following settings under each Tab heading indicated below.

<table>
<thead>
<tr>
<th>Tab</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Authentication Level should be set to “Default”.</td>
</tr>
<tr>
<td>Location</td>
<td>Normally, none of these options are selected, and the name of the computer on which to run the application is specified in the remote program (see the <strong>Connect</strong> (see page 118) object’s <strong>Instrument</strong> (see page 265) property). However, the ”Run application on the following computer” option can be checked, with the logic analyzer’s computer name entered in the field that follows.</td>
</tr>
<tr>
<td>Security</td>
<td>Use default launch (and activation if on Windows XP) permissions. Use default access permissions. Use Custom configuration permissions. Verify that you have “Full Control”.</td>
</tr>
<tr>
<td>Endpoints</td>
<td>Leave at default system protocols.</td>
</tr>
<tr>
<td>Identity</td>
<td>Normally, this tab does not appear (but it can if the Agilent Logic Analyzer application has been previously installed on the remote computer). If this tab appears, set to ”The interactive user”.</td>
</tr>
</tbody>
</table>

4 In the Agilent 168x/169x/169xx Logic Analyzer Properties dialog, click OK.
5 Close the Component Services window.
3 Using COM Automation

To programmatically control the logic analyzer via COM automation, you can use the integrated Microsoft Visual Basic for Applications (VBA) or you can install some other COM aware client software package like Visual Basic, Visual C++, LabVIEW, VEE, etc.

- Using Visual Basic for Applications (VBA) in Microsoft Excel (see page 42)
- Using Visual Basic (in Visual Studio) (see page 45)
- Example Visual Basic Programs (see page 46)
- Using Visual C++ (see page 70)
- Using LabVIEW (see page 72)
- Using Perl (see page 76)
- Using Python (see page 80)
- Using Tcl (see page 84)
Using Visual Basic for Applications (VBA) in Microsoft Excel

1. Install the Microsoft Excel software.
2. Import the type library:
   a. In the Visual Basic Editor, choose the **Tools>References...** menu item.
   b. In the References dialog, select the library "Agilent 168x/169x/169xx Logic Analyzer Object Library".
3. In the Excel Visual Basic Editor, copy and paste the GetLAData() code below. (In Excel 2000: Execute **Tools>Macro>Macros...** In the Macro Name box, type in "GetLAData". Then, press the **Create** button.)
4. Optionally, call the GetLAData() macro from a custom toolbar button, and watch the Worksheet update with the logic analysis data.

**Example**

```vba
Sub GetLAData()
    ' This Excel macro example transfers all of the bus/signal's
    ' from the first module in a 168x/9x/9xx Logic Analysis System to
    ' the Active Excel Worksheet. Variables to modify are:
    ' myInst   -> change to the hostname or IP address of the LA
    '          you're connecting to (default is 'localhost'
    '          if you're running Excel directly on the LA)
    ' mySheet  -> change the worksheet to copy the data to
    '          (default is the active worksheet)
    ' myAnalyzer-> change to the analyzer name to transfer data from
    '          (default is the first module)
    ' myStartSample, myEndSample -> change to the data range to upload
    '          (default is -10 and 10 respectively)
    
    ' Get the active Excel worksheet
    Dim mySheet As Worksheet
    Set mySheet = ActiveWorkbook.ActiveSheet

    ' Clear all of the cells in the worksheet
    mySheet.Cells.ClearContents

    ' Create the 168x/9x/9xx Logic Analyzer Instrument object
    ' and connect to the Logic Analyzer
    Dim myConnect As AgtLA.Connect
    Dim myInst As AgtLA.Instrument
    Set myConnect = CreateObject("AgtLA.Connect")
    Set myInst = myConnect.Instrument("localhost")

    ' Run the measurement, wait for completion or time out
    myInst.Run
    myInst.WaitComplete (10)

    ' Get the first analyzer module
```

3   Using COM Automation

42   COM Automation Online Help
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.Modules(0)

' Upload a range of acquired data and copy to the Excel worksheet
Dim myBusSignal As AgtLA.BusSignal
Dim myData As AgtLA.SampleBusSignalData

Dim myNumDataRows As Long
Dim myStartSample As Long
Dim myEndSample As Long

colNum = 1 ' Start putting the data in the first column
myStartSample = -10 ' Sample data range to upload
myEndSample = 10

' Copy over all bus/signals
For Each myBusSignal In myAnalyzer.BusSignals
    Set myData = myBusSignal.BusSignalData
    mySheet.Cells(1, colNum) = myBusSignal.Name
    Select Case myBusSignal.BusSignalType
        Case AgtBusSignalSampleNum
            Dim lArray() As Long
            lArray = myData.GetDataBySample(myStartSample, myEndSample, _
                AgtDataLong, myNumDataRows)
            For rowNum = 0 To myNumDataRows - 1
                mySheet.Cells(rowNum + 2, colNum) = lArray(rowNum)
            Next rowNum
        Case AgtBusSignalTime
            Dim dArray() As Double
            dArray = myData.GetDataBySample(myStartSample, myEndSample, _
                AgtDataTime, myNumDataRows)
            For rowNum = 0 To myNumDataRows - 1
                mySheet.Cells(rowNum + 2, colNum) = dArray(rowNum)
            Next rowNum
        Case AgtBusSignalGenerated
            Dim vArray As Variant ' Decimal holds max 96 bits unsigned.
            vArray = myData.GetDataBySample(myStartSample, myEndSample, _
                AgtDataDecimal, myNumDataRows)
            For rowNum = 0 To myNumDataRows - 1
                mySheet.Cells(rowNum + 2, colNum) = vArray(rowNum)
            Next rowNum
        Case AgtBusSignalProbed
            ' Long holds a maximum of 31 bits unsigned.
            lArray = myData.GetDataBySample(myStartSample, myEndSample, _
                AgtDataLong, myNumDataRows)
            For rowNum = 0 To myNumDataRows - 1
                ' format has hex for display purposes
                mySheet.Cells(rowNum + 2, colNum) = Hex$(lArray(rowNum))
            Next rowNum
    End Select
Next myBusSignal

' Go to the next bus/signal
colNum = colNum + 1
Next
End Sub

**See Also**
- Example Visual Basic Programs (see page 46)
Using Visual Basic (in Visual Studio)

Before you can use the Visual Basic programming environment to control the Agilent Logic Analyzer application, you must first import the Instrument Automation Server's type library into your project.

1. Choose the Project>References... menu item.
2. In the References dialog, select the library "Agilent 168x/169x/169xx Logic Analyzer Object Library".

See Also

- Example Visual Basic Programs (see page 46)
Example Visual Basic and Visual C++ Programs

- Loading, Running, Storing (see page 46)
- Setting Up Simple Triggers (see page 51)
- Setting Up Advanced Triggers (see page 54)
- Changing the Sampling Mode (see page 58)
- Checking the Logic Analyzer Software Version (see page 62)

See Also
- Additional Visual Basic Examples (see page 69)
- Using Visual Basic for Applications (VBA) in Microsoft Excel (see page 42)
- Using Visual Basic (in Visual Studio) (see page 45)

Loading, Running, Storing

In order to create an easy to use, yet powerful remote control mechanism, the design of the COM Automation Server adheres to the basic use model of "load-run-store".

In other words, to create a remote control application or a program that runs repetitive tests:

1. Use the Agilent Logic Analyzer application to go through each test once, and save the logic analyzer configurations and trigger setup specifications to files.

2. Then, from your program, load the appropriate logic analyzer configuration and trigger setup specification files, run the measurement, and store or act on the results as appropriate.

Example

If you encounter any name collisions (in other words, if you already have an object defined that uses the same name as an object in the Instrument Automation Server library), you can use the "AgtLA" library name prefix to resolve the conflict. For example, if you have a "Module" object defined, you can use "AgtLA.Module" to refer to the Instrument Automation Server's "Module" object.

Visual Basic

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the online help),
' the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst"
' with "AgtLA" to access the global Instrument object in VBA.
' Load the configuration file.
myInst.Open ("c:\LA\Configs\mpc860_demo_compare.ala")

' Load the logic analyzer trigger file.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.GetModuleByName("My 1690A-1")
myAnalyzer.RecallTriggerByFile ("c:\LA\Triggers\TrigSpecFile.xml")

' Run the measurement, wait for it to complete.
myInst.Run
myInst.WaitComplete (20)

' Process/display/store captured data.
Dim myBusSignal As AgtLA.BusSignal
Dim myData As AgtLA.SampleBusSignalData

For Each myBusSignal In myAnalyzer.BusSignals
    ' Get Data from "ADDR".
    If myBusSignal.Name = "ADDR" Then
        Set myData = myBusSignal.BusSignalData
        'Upload a range of acquired data.
        Dim myArray() As Long  ' The size is defined in GetDataBySample.
        Dim NumRows As Long
        myArray = myData.GetDataBySample(-10, 10, AgtDataLong, NumRows)
        ' Find the largest bus/signal value.
        Dim LongValue As Long
        Dim LargestValue As Long
        LargestValue = 0
        For i = 0 To NumRows - 1
            LongValue = myArray(i)
            If LongValue > LargestValue Then
                LargestValue = LongValue
            End If
        Next i
        MsgBox "Largest value is: " + Str(LargestValue)
    End If
Next

Visual C++

// This simple Console application demonstrates how to use the
// Agilent 168x/9x/9xx COM interface from Visual C++.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO")
//
// To run, you need to specify the host Logic Analyzer to connect
// to (search for "TODO")
//
#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.

    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // myLAHostname.
            printf("Connecting to instrument '%s'\n", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\LA\Configs\config.ala";
            printf("Loading the config file '%s'\n", (char*) configFile);
            pInst->Open(configFile, FALSE, "", TRUE);

            // Get a specific analyzer module.
            _bstr_t moduleName = "MPC860 Demo Board";
            AgtLA::IAnalyzerModulePtr pAnalyzer =
                pInst->GetModuleByName(moduleName);

            // Load the logic analyzer trigger file.
            _bstr_t triggerFile = "C:\LA\Configs\trigger.xml";
            printf("Loading the trigger file '%s'\n", (char*) triggerFile);
            pAnalyzer->RecallTriggerByFile(triggerFile);

            // Run the measurement, wait for it to complete.
            pInst->Run(FALSE);
            pInst->WaitComplete(20);

            // Process/display/store captured data.
            _bstr_t busSignal;
using _variant_t varArray;
long numRowsRet;
long numBytesPerRow;

AgtLA::IBusSignalsPtr pBusSignals = pAnalyzer->GetBusSignals();
for (long i = 0; i < pBusSignals->GetCount(); i++)
{
    busSignal = pBusSignals->GetItem(i)->GetName();

    // Get data from "ADDR" bus.
    if (strcmp(busSignal, "ADDR") == 0)
    {
        long numSamples;
        long lBound;

        AgtLA::IBusSignalPtr pBusSignal =
            pAnalyzer->GetBusSignals() - GetItem(busSignal);
        AgtLA::ISampleBusSignalDataPtr pSampleData =
            pBusSignal->GetBusSignalData();
        varArray = pSampleData->GetDataBySample(-10, 10,
            AgtLA::AgtDataRaw, &numRowsRet);
        numBytesPerRow = pBusSignal->GetByteSize();
        HRESULT hr = SafeArrayGetLBound(varArray.parray, 1,
            &lBound);

        if (SUCCEEDED(hr))
        {
            long uBound;
            hr = SafeArrayGetUBound(varArray.parray, 1, &uBound);

            if (SUCCEEDED(hr))
            {
                byte* pByteArray;
                hr = SafeArrayAccessData(varArray.parray,
                    (void**) &pByteArray);

                if (SUCCEEDED(hr))
                {
                    numSamples =
                        (uBound - lBound + 1) / numBytesPerRow;
                    byte* pByte = pByteArray;
                    printf("Displaying 's ADDR bus samples from -10 ");
                    printf("to 10:\n");

                    for (int i = 0; i < numSamples; i++)
                    {
                        printf(" sample[%d]: ", -10 + i);

                        for (int j = 0; j < numBytesPerRow; j++)
                        {
                            printf("%02x ", pByte[j]);
                        }

                        pByte += numBytesPerRow;
                        printf("\n");
                    }
                }
            }
        }
    }
}
printf("\n");
SafeArrayUnaccessData(varArray.parray);
}
}
}
}
}
}
}
}
}
}
}
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
}
else {
    printf("CoInitialize failed\n");
}

return 0;
}

//////////////////////////////////////////////////////////////////////

// Displays the last error -- used to show the last exception
// information.
//
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");

    printf("Fatal Unexpected Error: \n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0) {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }
printf(" Error Message = %s\n", (char*) errorStr);
}

Setting Up Simple Triggers

This example shows how to set up simple triggers using the SimpleTrigger (see page 232) method of the AnalyzerModule (see page 106) object.

Visual Basic

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Load the configuration file.
myInst.Open("c:\LA\Configs\mpc860_demo_compare.ala")

' Declare trigger variables.
Dim mySimpleTriggers(2) As String
mySimpleTriggers(0) = "ADDR=hfff034d8"
mySimpleTriggers(1) = "ADDR=h00004088 And DATA=h46xxxxxx"
mySimpleTriggers(2) = "ADDR=h000041ad And DATA=h47xxxxxx"
Dim I As Integer

' Set up triggers using the SimpleTrigger method.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.GetModuleByName("My 1690A-1")

For I = 0 To 2
    myAnalyzer.SimpleTrigger mySimpleTriggers(I)

    ' Run the measurement, wait for it to complete.
    myInst.Run
    myInst.WaitComplete (20)

    ' Process/display/store captured data.
Next

Visual C++

// This simple Visual C++ Console application demonstrates how to use simple triggers with the Agilent 168x/9x/9xx COM interface.

// This project was created in Visual C++ Developer. To create a similar project:
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:\Program Files\Agilent Technologies\Logic Analyzer\LA \nCOM Automation\agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[]) {  
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr)) {
        try {  // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33";  // TODO, use your logic
                                        // analysis system hostname.
            printf("Connecting to instrument '%s'\n", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\LA\Configs\config.ala";
            printf("Loading the config file '%s'\n", (char*) configFile);
            pInst->Open(configFile, FALSE, "", TRUE);

            // Declare trigger variables.
            _bstr_t mySimpleTriggers[] = {
                "ADDR=hfff034d8",
                "ADDR=h00004088 And DATA=h46xxxxxx",
                "ADDR=h000041ad And DATA=h47xxxxxx"
            };
        } catch (_com_error& err) {  // Catch any unexpected run-time errors.
            DisplayError(err);
        }
    }

    return 0;
}

COM Automation Online Help
// Set up triggers using the SimpleTrigger method.
_bstr_t moduleName = "MPC860 Demo Board";
AgtLA::IAnalyzerModulePtr pAnalyzer =
pInst->GetModuleByName(moduleName);
for (long i = 0; i < 3; i++)
{
    printf("Trigger when '%s' occurs once, store anything.\n", (char*) mySimpleTriggers[i]);
pAnalyzer->SimpleTrigger(mySimpleTriggers[i], 1,
    "Anything");

    // Run the measurement, wait for it to complete.
pInst->Run(FALSE);
pInst->WaitComplete(20);

    // Process/display/store captured data.
}
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
else
{
    printf("CoInitialize failed\n");

    return 0;
}

VERRIDE

// Displays the last error -- used to show the last exception
// information.

void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");

    printf("Fatal Unexpected Error: \n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
The SimpleTrigger (see page 232) method cannot set complex, multiple-step trigger sequences. To do that, you must set the AnalyzerModule (see page 106) object's Trigger (see page 285) property to an XML-format trigger specification string.

See Also
• Setting Up Advanced Triggers (see page 54)

Setting Up Advanced Triggers

This example shows how to set up advanced triggers by setting the Trigger (see page 285) property of the AnalyzerModule (see page 106) object to an XML-format trigger specification string.

Visual Basic
' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Load the configuration file.
myInst.Open ("c:\LA\Configs\mpc860_demo_compare.ala")

' Declare trigger variables.
Dim myTriggerFiles(2) As String
myTriggerFiles(0) = "c:\LA\Triggers\TrigSpecFile0.xml"
myTriggerFiles(1) = "c:\LA\Triggers\TrigSpecFile1.xml"
myTriggerFiles(2) = "c:\LA\Triggers\TrigSpecFile2.xml"
Dim I As Integer

' Set triggers using the logic analyzer Trigger property.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.GetModuleByName("My 1690A-1")
Dim myTrigger As String
Dim myTrigFileNum
myTrigFileNum = FreeFile

For I = 0 To 2
' Get trigger spec. from local file.
Open myTriggerFiles(I) For Input As myTrigFileNum
' InputB copies bytes from a file into a variable.
' StrConv converts the ANSI string to a UNICODE string.
myTrigger = StrConv(InputB(LOF(myTrigFileNum), myTrigFileNum), vbUnicode)
Close myTrigFileNum

' Set up the logic analyzer trigger.
myAnalyzer.Trigger = myTrigger

' Or, to get trigger spec. from file on the
' instrument (and set the Trigger property):
'myAnalyzer.RecallTriggerByFile (myTriggerFiles(I))

' Display the logic analyzer trigger specification.
myTrigger = myAnalyzer.Trigger
MsgBox myTrigger

' Run the measurement, wait for it to complete.
myInst.Run
myInst.WaitComplete (20)

' Process/display/store captured data.
Next

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// advanced triggers with the Agilent 168x/9x/9xx COM interface.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <sstream>
Using COM Automation

```c
void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
            printf("Connecting to instrument '%s'\n", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\LA\Configs\config.ala";
            printf("Loading the config file '%s'\n", (char*) configFile);
            pInst->Open(configFile, FALSE, ",", TRUE);

            // Declare trigger variables.
            _bstr_t myTriggerFiles[] = {
                "c:\LA\Triggers\TrigSpecFile0.xml",
                "c:\LA\Triggers\TrigSpecFile1.xml",
                "c:\LA\Triggers\TrigSpecFile2.xml"
            };

            // Set up triggers using the SimpleTrigger method.
            _bstr_t moduleName = "MPC860 Demo Board";
            AgtLA::IAnalyzerModulePtr pAnalyzer =
                pInst->GetModuleByName(moduleName);
            for (long i = 0; i < 3; i++)
            {
                _bstr_t myTriggerSpec;

                // Get trigger spec. from local file.
                std::wifstream inFile(myTriggerFiles[i]);
                std::wstringstream inBuffer; // Intermediate buffer.
                inBuffer << inFile.rdbuf(); // Read entire file.
                // Create bstr.
                myTriggerSpec = SysAllocString(inBuffer.str().c_str());

                // Set up the logic analyzer trigger.
```
printf("Loading local trigger file '%s'\n", (char*) myTriggerFiles[i]);
pAnalyzer->PutTrigger(myTriggerSpec);

// Or, to load trigger spec. from instrument (and set the
// Trigger property):
//printf("Loading trigger from instrument '%s'\n", (char*) myTriggerFiles[i]);
//pAnalyzer->RecallTriggerByFile(myTriggerFiles[i]);

// Display the logic analyzer trigger specification.
myTriggerSpec = pAnalyzer->GetTrigger();
printf("XML trigger spec: '%s'\n", (char*) myTriggerSpec);

// Run the measurement, wait for it to complete.
pInst->Run(FALSE);
pInst->WaitComplete(20);

  // Process/display/store captured data.
}
}
catch (_com_error& e) {  
  DisplayError(e);
}

  // Uninitialize the Microsoft COM/ActiveX library.
  CoUninitialize();
} else  
{
    printf("CoInitialize failed\n");
}

  return 0;
}

////////////////////////////////////////////////////////////////////////
//
// Displays the last error -- used to show the last exception
// information.
//
void DisplayError(_com_error& error)
{
  printf("*** DisplayError()\n");

  printf("Fatal unexpected Error:\n");
  printf(" Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();
  if (desc.length() == 0) {
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

Using COM Automation

```c
// Remove funny carriage return ctrl<M>
if (nl>2 && (errorStr[nl-2] == 0xd))
{
    errorStr[nl-2] = '\0';
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);
```

See Also  
- Setting Up Simple Triggers (see page 51)

Changing the Sampling Mode

This example shows how to change the logic analyzer sampling mode by using XML-format strings with the Setup (see page 277) property of the AnalyzerModule (see page 106) object.

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it ' to access the Instrument object. In this example, "myInst" ' represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the online help), ' the Instrument object is already created and is globally ' accessible using "AgtLA". In this example, substitute "myInst" ' with "AgtLA" to access the global Instrument object in VBA.

' Create the logic analyzer object.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.GetModuleByName("My 1690A-1")
Dim mySetup As String

' Set the timing (asynchronous) sampling mode.
Dim myTimingSamplingSetup As String
myTimingSamplingSetup = "<Module>" + _
    "<SamplingSetup>" + _
    "<Sampling ChannelMode='Full' MaxSpeed='400' " + _
    "SamplePeriod='2.5 ns' Type='Standard' Acquisition='Timing' " + _
    "AcquisitionDepth='256K' TriggerPosition='50'/>" + _
    "</SamplingSetup>" + _
"</Module>"
myAnalyzer.Setup = myTimingSamplingSetup

' Display the complete logic analyzer setup.
mySetup = myAnalyzer.Setup
MsgBox mySetup

' Set the state (synchronous) sampling mode.
Dim myStateSamplingSetup As String
myStateSamplingSetup = "<Module>" + _
   "<SamplingSetup>" + _
   "<Sampling ChannelMode='Full' Acquisition='State' " + _
      "AcquisitionDepth='256K' MaxSpeed='200' " + _
      "TriggerPosition='50'/>" + _
   "<StateClockSpec Mode='Master'>" + _
      "<Clear/>" + _
      "<Master>" + _
      "<ClockGroup>" + _
         "<Edges>" + _
            "<Edge PodIndex='1' Value='Rising'/>" + _
         "</Edges>" + _
      "<Qualifiers Operator='And'>" + _
         "<Qualifier Level='Low' PodIndex='2'/>" + _
      "</Qualifiers>" + _
      "</ClockGroup>" + _
   "</Master>" + _
   "</StateClockSpec>" + _
   "</SamplingSetup>" + _
"</Module>
myAnalyzer.Setup = myStateSamplingSetup

' Display the complete logic analyzer setup.
mySetup = myAnalyzer.Setup
MsgBox mySetup

Visual C++

// This simple Visual C++ Console application demonstrates
// how to change the logic analyzer sampling mode with the
// Agilent 168x/9x/9xx COM interface.

// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \COM Automation/agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect
to (search for "TODO" below).

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);
main() entry point.

```c
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            printf("Connecting to instrument '%s'\n", (char*)hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Get the logic analyzer object.
            _bstr_t moduleName = "My 16910A-1";
            AgtLA::IAnalyzerModulePtr pAnalyzer =
                pInst->GetModuleByName(moduleName);

            // Set the timing (asynchronous) sampling mode.
            _bstr_t myTimingSamplingSetup="
                <Module> 
                    <SamplingSetup> 
                        <Sampling ChannelMode='Full' MaxSpeed='400' 
                            SamplePeriod='2.5 ns' Type='Standard' 
                            Acquisition='Timing' AcquisitionDepth='256K' 
                            TriggerPosition='50'/>
                    </SamplingSetup>
                </Module>
            ";
            pAnalyzer->PutSetup(myTimingSamplingSetup);

            // Display the complete logic analyzer setup.
            _bstr_t mySetup;
            mySetup = pAnalyzer->GetSetup();
            printf("Logic analyzer setup: '%s'\n", (char*)mySetup);

            // Set the state (synchronous) sampling mode.
            _bstr_t myStateSamplingSetup="
                <Module> 
                    <SamplingSetup> 
                        <Sampling ChannelMode='Full' Acquisition='State' 
                            AcquisitionDepth='256K' MaxSpeed='200' 
                            TriggerPosition='50'/>
                    </SamplingSetup>
                </Module>
            ";
        }
    }
}
```
<StateClockSpec Mode='Master'>
  <Clear/>
  <Master>
    <ClockGroup>
      <Edges>
        <Edge PodIndex='1' Value='Rising'/>
      </Edges>
      <Qualifiers Operator='And'>
        <Qualifier Level='Low' PodIndex='2'/>
      </Qualifiers>
    </ClockGroup>
  </Master>
</StateClockSpec>
</SamplingSetup>
}
pAnalyzer->PutSetup(myStateSamplingSetup);

// Display the complete logic analyzer setup.
mySetup = pAnalyzer->GetSetup();
printf("Logic analyzer setup: '%s'", (char*) mySetup);

} catch (_com_error& e) {
  DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
}
else {
  printf("CoInitialize failed\n");
}

return 0;
}

/////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.

void DisplayError(_com_error& error)
{
  printf("*** DisplayError()\n");

  printf("Fatal Unexpected Error:\n");
  printf(" Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();

  if (desc.length() == 0) {
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
  } else {
    // Tack on the description string.
    int errorLen = desc.length();
    desc.Delete(0, errorLen + 1);
    // Append the description.
    const char* descPtr = desc.BStr();
    for (int i = 0; i < errorLen; i++) {
      errorStr[i] = descPtr[i];
    }
  }

  printf("Error Description: \n");
  printf("%s\n", errorStr);

  printf("Error Source: \n");
Using COM Automation

int nLen = lstrlen(errorStr);

// Remove funny carriage return ctrl\M.
if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
{
    errorStr[nLen - 2] = '\0';
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);

Checking the Logic Analyzer Software Version

This example shows you how to check for the correct logic analyzer software version before using recently added objects, methods, and properties.

Visual Basic

Public Sub CheckVersion()
    Dim strVersion As String
    strVersion = AgtLA.VBAVersion
    bResult = DoesCurrentVersionMatchRequiredVersion("03.06.01")
End Sub

Private Function DoesCurrentVersionMatchRequiredVersion(ByVal _
    strRequiredVersion As String) As Boolean

    ' Get the version number, broken down into:
    ' xx.yyy.zzz
    ' where xx is the major version
    ' yy is the minor version
    ' zz is the SubMinor version (which may not be present)
    ' Example :
    ' 03.00.01 has major version 3, minor version 0, sub-minor version 1
    ' If the Required Version is 03.00 and the current VBA Version is
    ' 02.80, then false is returned. But, if the VBA Version is 03.01,
    ' then true is returned.

    Dim nRequiredMajor As Integer
    Dim nRequiredMinor As Integer
    Dim nRequiredSubMinor As Integer
    Dim nCurrentMajor As Integer
    Dim nCurrentMinor As Integer
    Dim nCurrentSubMinor As Integer

    ' If the two versions are identical, we're done.
    If (strRequiredVersion = AgtLA.Version) Then
DoesCurrentVersionMatchRequiredVersion = True
Exit Function
End If

Call GetNumbersForVersionString(strRequiredVersion, _
    nRequiredMajor, _
    nRequiredMinor, _
    nRequiredSubMinor)

Call GetNumbersForVersionString(AgtLA.Version, _
    nCurrentMajor, _
    nCurrentMinor, _
    nCurrentSubMinor)

' Check Major Version first.
If (nCurrentMajor > nRequiredMajor) Then
    DoesCurrentVersionMatchRequiredVersion = True
    Exit Function
Else
    If (nCurrentMajor < nRequiredMajor) Then
        DoesCurrentVersionMatchRequiredVersion = False
        Exit Function
    End If
End If

' Check Minor Version.
If (nCurrentMinor > nRequiredMinor) Then
    DoesCurrentVersionMatchRequiredVersion = True
    Exit Function
Else
    If (nCurrentMinor < nRequiredMinor) Then
        DoesCurrentVersionMatchRequiredVersion = False
        Exit Function
    End If
End If

' Check SubMinor Version.
If (nCurrentSubMinor > nRequiredSubMinor) Then
    DoesCurrentVersionMatchRequiredVersion = True
    Exit Function
Else
    If (nCurrentSubMinor < nRequiredSubMinor) Then
        DoesCurrentVersionMatchRequiredVersion = False
        Exit Function
    End If
End If

End Function

Private Sub GetNumbersForVersionString(ByVal strVersion As String, _
    ByRef nMajor As Integer, _
    ByRef nMinor As Integer, _
    ByRef nSubMinor As Integer)

    Dim nDash As Integer
    Dim nFirstPeriod As Integer
    Dim nSecondPeriod As Integer

On Error GoTo invalidStr
' If there's a dash, eliminate.
  nDash = InStr(1, strVersion, "-"
If (nDash > 0) Then
  strVersion = Mid(strVersion, 1, nDash - 1)
End If

' Get the Version first. Put up a message if the string is wrong.
  nFirstPeriod = InStr(1, strVersion, ".")

' If there's no period, we need to exit.
If (nFirstPeriod = 0) Then
  MsgBox "The version string " + strVersion + ", is not valid. Examples are 03.02.01 or 03.00."
  Exit Sub
End If

nSecondPeriod = InStr(nFirstPeriod + 1, strVersion, ".")
nMajor = CInt(Mid(strVersion, 1, nFirstPeriod - 1))
If (nSecondPeriod = 0) Then
  nMinor = CInt(Mid(strVersion, nFirstPeriod + 1, _
    Len(strVersion) - nFirstPeriod))
nSubMinor = 0
Else
  nMinor = CInt(Mid(strVersion, nFirstPeriod + 1, _
    nSecondPeriod - nFirstPeriod - 1))
nSubMinor = CInt(Mid(strVersion, nSecondPeriod + 1, _
    Len(strVersion) - nSecondPeriod))
End If

Exit Sub

invalidStr:
  MsgBox "The version string " + strRequiredVersion + " + _
    " is not valid. Examples are 03.02.01 or 03.00."
End Sub

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to check the system
// software version.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// instdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
```cpp
#include "stdafx.h"
#include <string>
using namespace std;

// Forward declarations.

// GetNumbersForVersionString:
//_bstr_t& strVersion,
//long& nMajor,
//long& nMinor,
//long& nSubminor);}

// DoesCurrVersionMatch:
//_bstr_t& strReqdVersion);

void DisplayError(_com_error& err);

int main(int argc, char* argv[]) {
    printf("*** Main()\n");
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr)) {
        _bstr_t strReqdVersion = "03.00.0001";
        if (DoesCurrVersionMatch(strReqdVersion)) {
            printf("Current version matches required '%s'\n", (char*) strReqdVersion);
        } else {
            printf("Current version does not match required '%s'\n", (char*) strReqdVersion);
        }

        // Uninitialize the Microsoft COM/ActiveX library.
        CoUninitialize();
    } else {
        printf("CoInitialize failed\n");
    }
```
return 0;
}

// Given a version string, return major, minor, and subminor numbers.
boolean GetNumbersForVersionString(_bstr_t& version,
      long& nMajor,
      long& nMinor,
      long& nSubminor)
{
    string strVersion = version;
    string strMajor;
    string strMinor;
    string strSubminor;

    string::size_type nDash;
    string::size_type nFirstPeriod;
    string::size_type nSecondPeriod;

    // If there's a dash, eliminate from dash to end of string.
    nDash = strVersion.find("-");
    if (nDash != string::npos) {
      strVersion = strVersion.substr(0, nDash);
    }

    // Get the Version first. Put up a message if the string is wrong.
    nFirstPeriod = strVersion.find(".");
    if (nFirstPeriod == string::npos) {
      printf("The version string '%s' is not valid.",
           strVersion.c_str());
      printf(" Examples are 03.02.01 or 03.00.\n");
      return (false);
    }
    strMajor = strVersion.substr(0, nFirstPeriod);
    nMajor = atol(strMajor.c_str());
    nSecondPeriod = strVersion.find(".", nFirstPeriod + 1);
    if (nSecondPeriod == string::npos) { // No second period.
      strMinor = strVersion.substr(nFirstPeriod+1);
      nMinor = atol(strMinor.c_str());
      strSubminor = ""
      nSubminor = 0;
    } else { // There is a second period.
      strMinor = strVersion.substr(nFirstPeriod+1,
                                   nSecondPeriod-nFirstPeriod-1);
      nMinor = atol(strMinor.c_str());
      strSubminor = strVersion.substr(nSecondPeriod+1);
      nSubminor = atol(strSubminor.c_str());
    }
}
return (true);
}

//////////////////////////////////////////////////////////////////////
//
// Returns true if the current system software version matches
// the required version.
//
boolean DoesCurrVersionMatch(_bstr_t& strReqdVersion)
{
    // Get the version number, broken down into:
    // xx.yyy.zzz
    // where xx is the major version
    // yy is the minor version
    // zz is the Subminor version (which may not be present)
    //
    // Example:
    // 03.00.01 has the major version 3, minor version 0, sub-minor
    // version 1
    //
    // If the Required Version is 03.00 and the current VBA Version is
    // 02.80, then false is returned. But, if the VBA Version is 03.01,
    // then true is returned.
    
    long nReqdMajor;
    long nReqdMinor;
    long nReqdSubminor;
    long nCurrMajor;
    long nCurrMinor;
    long nCurrSubminor;

    try { // Catch any unexpected run-time errors.
        _bstr_t hostname = "mtx33"; // TODO, use your logic analysis
                                    // system hostname.
        printf("Connecting to instrument \"%s\n", (char*) hostname);

        // Create the connect object and get the instrument object.
        AgtLA::IConnectPtr pConnect =
            AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
        AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostname);

        // Get current system software version.
        _bstr_t strCurrVersion = pInst->GetVersion();

        // If the two versions are identical, we're done.
        if (strReqdVersion == strCurrVersion) {
            return (true);
        }

        // Get the individual numbers for the version string.
        if (!GetNumbersForVersionString(strReqdVersion, nReqdMajor,
                                         nReqdMinor, nReqdSubminor)) {
            return (false);
        }

        if (!GetNumbersForVersionString(strCurrVersion, nCurrMajor,
nCurrMinor, nCurrSubminor)) {
    return (false);
}

// Check Major version first.
if (nCurrMajor > nReqdMajor) {
    return (true);
} else {
    if (nCurrMajor < nReqdMajor) {
        return (false);
    }
}

// Check Minor version next.
if (nCurrMinor > nReqdMinor) {
    return (true);
} else {
    if (nCurrMinor < nReqdMinor) {
        return (false);
    }
}

// Check Subminor version last.
if (nCurrSubminor > nReqdSubminor) {
    return (true);
} else {
    if (nCurrSubminor < nReqdSubminor) {
        return (false);
    }
}

// All numbers the same, return true.
return (true);
}
catch (_com_error& e) {
    DisplayError(e);
    return (false);
}
}

/* Displays the last error -- used to show the last exception information. */
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\", error.Error());
    static char errorStr[1024];
_bstr_t desc = error.Description();

if (desc.length() == 0)
{
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
    {
        errorStr[nLen - 2] = '\0';
    }
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);

**Additional Visual Basic Examples**

Visual Basic example projects can be found in your install directory. The default installation example directory is C:/Program Files/Agilent Technologies/Logic Analyzer/LA COM Automation/Visual Basic Examples.

Before running each example, read the documentation at the top of each source file for an explanation of how the logic analyzer must be set up before the example will run successfully.
Using Visual C++

This online help is geared mainly towards Visual Basic not Visual C++ programmers. Although you'll find the documentation is not directly applicable, the explanation of object, methods, properties and parameters can still be helpful.

Visual C++ Examples

Visual C++ example projects can be found in your install directory. The default installation example directory is C:/Program Files/Agilent Technologies/Logic Analyzer/LA COM Automation/Visual C++ Examples.

Before running each example, read the documentation at the top of each source file for an explanation of how the logic analyzer must be set up before the example will run successfully.

Simple Visual C++ Example

This example connects to the logic analyzer hardware and starts a measurement.

This C++ example uses COM smart pointers _comptr_t to integrate the ActiveX/COM automation server into the Visual C++ environment. The declaration of the Instrument specific smart pointers is in the "agClientSvr.tlh" header file which is automatically created and included when the instrument type library is imported using the #import directive.

For detailed method and property parameter types, see the "agClientSvr.tlh" header file. Note that Get/Put methods are generated for properties that you can get or set (as described in this online help). For example, the GetInstrument method in the following example accesses the Instrument (see page 265) property.

Using smart pointers is not the only way to integrate COM into the Visual C++ environment. For more details, refer to the Microsoft Visual C++ documentation.

```c++
// Import the Instrument Automation Server's type library.
// Replace <install_dir> with your installation directory.
// Default is:
// C:/Program Files/Agilent Technologies/Logic Analyzer/LA COM Automation/
#import "<install_dir>/agClientSvr.dll"

// Before using the Automation Server, initialize the COM/OLE libraries in your MFC application's InitInstance() method.
if (!AfxOleInit())
{
    ::AfxMessageBox("OLE initialization failed");
    return FALSE;
}

// If you're not using MFC, you should call
//CoInitialize(0)/CoUninitialize()
```
// For detailed method and property parameter types,  
// see the header file "agClientSvr.tlh generated automatically  
// by the #import directive in your project's configuration directory.

// Place the following C++ code in your class method.  
try {
  // create the connect object and get the instrument  
  AgtLA::IConnectPtr pConnect =  
      AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));  
  AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument("");  
  // run all modules
  pInst->Run(FALSE);
}

catch (_com_error& e) {
  // Display any error messages returned.
  _bstr_t msg(e.Description());
  if (msg.length() == 0)  
    msg = e.ErrorMessage();
  ::AfxMessageBox(msg);
}
Using LabVIEW

- Tutorial - To programmatically control the logic analyzer in LabVIEW (see page 72)
- LabVIEW Examples (see page 75)

**Tutorial - To programmatically control the logic analyzer in LabVIEW**

Descriptions of the basic LabVIEW interface, its operation, and general use is not covered here. Refer to your LabVIEW online help for this information. The following steps are intended only as a guideline.

- LabVIEW 8.0 (see page 72)
- LabVIEW 6.0 (see page 73)

**LabVIEW 8.0**

1. Open the logic analyzer Connect object:
   a. In the Labview Front Panel, go to the Refnum and choose Automation Refnum.
   b. Right click select "ActiveX" class, choose browse.
   c. Select Object from Type library dialog, click/select "show creatable objects only" box.
   d. Choose "Agilent 168x/169x/169xx Logic Analyzer Object Library Version 1.0".
   e. Choose the "Connect(AgtLA.Connect.1)" object.

2. Get the logic analyzer Instrument Object:
   a. In the LabVIEW block diagram, go to the "Function" Palette and choose "Connectivity" then "ActiveX".
   b. Inside the "ActiveX" palette, drag the "Invoke Node" icon into the window.
   c. Press the "Connect Wire" button in the "Tools" Palette and connect the "Automation Open" icon's "Automation Refnum" output to the "Invoke Node" icon's "reference" input. The "Invoke Node" name displayed is now "IConnect".
   d. Right click on the "IConnect" icon and choose Methods->Instrument
   e. Connect a String Constant to the "HostNameOrIPAddress".
   f. The output of the "Instrument" method is an "IInstrument" object. Use this object to call the instrument's methods and properties.
3 Call a logic analyzer Instrument object method:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Invoke Node" icon onto the window.
   c Press the "Connect Wire" button in the "Tools" Palette and connect the "IConnect" icon's "Instrument" method output to the "Invoke Node" icon's "reference" input. The "Invoke Node" name displayed is now "IInstrument".
   d Right-click on the "IInstrument" icon and choose "Methods" to call any method.

4 Call a logic analyzer Instrument object property:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Property Node" icon onto the window.
   c Press the "Connect Wire" button in the "Tools" Palette and connect the "IConnect" icon's "Instrument" method output to the "Property Node" icon's "reference" input. The "Property Node" name displayed is now "IInstrument".
   d Right-click on the "IInstrument" icon and choose "Properties" to call any property.

LabVIEW 6.0

NOTE

The process documented here assumes you are using LabVIEW 6.0. This process may change for different versions of LabVIEW.

1 Open the logic analyzer Connect object:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Automation Open" icon onto the window.
   c Right-click on the "Automation Open" icon and choose "Select ActiveX Class"; then, choose "Browse...".
   d The Select Object From Type Library dialog will be displayed.
   e Choose "Agilent 168x/169x/169xx Logic Analyzer Object Library Version 1.0".
   f Choose the "Connect (AgtLA.Connect.1)" object.
2 Get the logic analyzer Instrument object:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Invoke Node" icon onto the window.
   c Press the "Connect Wire" button in the "Tools" Palette and connect the "Automation Open" icon's "Automation Refnum" output to the "Invoke Node" icon's "reference" input. The "Invoke Node" name displayed is now "IConnect".
   d Right click on the "IConnect" icon and choose Methods->Instrument
   e Connect a String Constant to the "HostNameOrIPAddress".
   f The output of the "Instrument" method is an "IInstrument" object. Use this object to call the instrument's methods and properties.

3 Call a logic analyzer Instrument object method:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Invoke Node" icon onto the window.
   c Press the "Connect Wire" button in the "Tools" Palette and connect the "IConnect" icon's "Instrument" method output to the "Invoke Node" icon's "reference" input. The "Invoke Node" name displayed is now "IInstrument".
   d Right-click on the "IInstrument" icon and choose "Methods" to call any method.

4 Call a logic analyzer Instrument object property:
   a In the LabVIEW diagram window, go to the "Function" Palette and choose "Communication", then "ActiveX".
   b Inside the "ActiveX" palette, drag the "Property Node" icon onto the window.
   c Press the "Connect Wire" button in the "Tools" Palette and connect the "IConnect" icon's "Instrument" method output to the "Property Node" icon's "reference" input. The "Property Node" name displayed is now "IInstrument".
   d Right-click on the "IInstrument" icon and choose "Properties" to call any property.

See the LabVIEW examples (see page 75) for a detailed view of how to use Invoke and Property Nodes.
LabVIEW Examples

LabVIEW examples can be found in your install directory. The default installation example directories are:

- C:/Program Files/Agilent Technologies/Logic Analyzer/LA COM Automation/LabVIEW 6.0 Examples
- C:/Program Files/Agilent Technologies/Logic Analyzer/LA COM Automation/LabVIEW 7.0 Examples
Using Perl

1. Install the Perl software.
2. Copy and paste the example code below into a file (PrintLAData.pl).
3. Run the example from the Command Prompt by entering the command: "perl PrintLAData.pl".

Example

```
# This Perl example prints all of the bus/signal's from the first module in a 168x/9x/9xx Logic Analysis System.
#
# $LAHostNameOrIP -> change to the hostname or IP address of the LA you're connecting to (default is 'localhost')
# if you're running Perl directly on the LA
# $LAAnalyzer -> change to the analyzer name to transfer data from (default is the first module)
# $LAStartRange, $LAEndRange -> change to the data range to upload (default is -10 and 10 respectively)
#
# This example was tested using ActiveState Perl version 5.6.1
#
# when using strict, declare *all* globals
use strict qw(vars refs subs);

# libraries needed to interface with the Logic Analyzer COM interface
use Win32::OLE;
use Win32::OLE::Variant;
use Win32::OLE::Const;

### ==================================================================
### * Begin Subroutines *
### ------------------------------------------------------------------
##--------------------------------------------------------------------
## FUNCTION:
## PrintArrays -- prints the Logic Analyzer Data Arrays
## SYNOPSIS:
## ARGUMENTS:
## arrays - array of data arrays to format and print. The first array
## contains the bus/signal names.
##--------------------------------------------------------------------

sub PrintArrays
{
    my @arrays = @_;

    my @nameArray = @{$arrays[0]};

    # Calculate the max width for each column and store in an array
    my @maxWidthArray;
```
print("\n");

for my $i ( 1 .. $#arrays )
{
    my $maxlen = length($nameArray[$i-1]);
    for my $j ( 0 .. $#{ $arrays[1] } )
    {
        my $data = $arrays[$i][$j];
        if (length($data) > $maxlen) {
            $maxlen = length($data);
        }
    }
    push(@maxWidthArray, $maxlen);
}

# Print the header row
for my $i ( 0 .. $#nameArray )
{
    my $hdr = $nameArray[$i];
    my $firstSpaces = ($maxWidthArray[$i] - length($hdr))/2;
    print " "." "x $firstSpaces;
    print $hdr;
    print " " x ($maxWidthArray[$i] - length($hdr) - $firstSpaces);
    print "\n";
}

for my $i ( 0 .. $#nameArray )
{
    print " "."-" x $maxWidthArray[$i];
    print "\n";
}

# Print the data rows
for my $i ( 0 .. $#{ $arrays[1] } )
{
    for my $j ( 1 .. $#arrays )
    {
        my $data = $arrays[$j][$i];
        print " "." "x ($maxWidthArray[$j-1] - length($data)) . $data;
    }
    print "\n";
}

### ------------------------------------------------------------------
### End of Subroutines
### ==================================================================
### ==================================================================
### * Begin Main Routine *
### ------------------------------------------------------------------
# Logic Analyzer host name or IP address
my $LAHostNameOrIP = "localhost";

# Create the logic analyzer client server
my $LAConnect = Win32::OLE->new('AgTLA.Connect');
if (! $LAConnect)
{
    print("Connection failed: ");
    print("please install the LA COM Automation client software\n");
    exit 1;
}

# Get the typedef constants
my $LAConstants = Win32::OLE::Const->Load($LAConnect);

# Connect to the remote logic analyzer instrument
print("\nConnecting to '$LAHostNameOrIP'\n");
my $LAInst = $LAConnect->Instrument($LAHostNameOrIP);
if (Win32::OLE->LastError != 0)
{
    print("Connection failed: ");
    print("please verify the LA hostname or IP address ");
    print("$LAHostNameOrIP\n");
    exit 1;
}

# Optionally load a configuration file that exists on the logic
# analyzer. If the file only exists on your client PC, then use the
# $LAConnect->CopyFile() method to copy the file onto your logic
# analyzer
#
# $LAInst->Open("test.xml");

# Run the analyzer and wait for the measurement to complete before
# getting data
$LAInst->Run();
$LAInst->WaitComplete(10);  # time out after 10 seconds

# Get the first module's data
my $LAAnalyzer = $LAInst->Modules(0);

# Get the module's bus/signal names and store into 'LANameArray'
#
my $LABusSignals = $LAAnalyzer->BusSignals();
my $LABusSignalsCount = $LABusSignals->count;
my @LANameArray;
my @LADataArrays;
my $numRows = Variant(VT_I4 | VT_BYREF, 0);
my $LAStartRange = -10;
my $LAEndRange = 10;

foreach my $index (0..$LABusSignalsCount-1)
{
    my $name = $LABusSignals->Item($index)->Name;
    push(@LANameArray, $name);
}
push (@LADataArrays, [@LANameArray]);

# Get the module's bus/signal data and store into 'LADataArrays'
foreach my $index (0..$LABusSignalsCount-1)
{
    my $LAData = $LABusSignals->Item($index)->BusSignalData;
    my $LABusSignalType = $LABusSignals->Item($index)->BusSignalType();
    my $LADataType = $LAConstants->{AgtDataLong};

    if ($LABusSignalType == $LAConstants->{AgtBusSignalTime})
    {
        $LADataType = $LAConstants->{AgtDataTime};
    }

    my $LADataArray = $LAData->GetDataBySample($LAStartRange,
        $LAEndRange, $LADataType, $numRows);
    push(@LADataArrays, $LADataArray);
}

# Print the Arrays
# PrintArrays(@LADataArrays);
Using Python

1. Install the Python and Python for Windows extension software.
2. Set up early binding for COM objects by running the MakePy utility. MakePy is a normal Python module that lives in the `win32com\client` directory of the PythonCOM package. There are two ways to run this script:
   - Start PythonWin, and from the Tools menu, select the item COM Makepy utility.
   - Using the Windows Explorer, locate the `client` subdirectory under the main `win32com` directory and double-click the file `makepy.py`.

In both cases, you are presented with a list of objects MakePy can use to support early binding. Select Agilent 168x/169x/169xx Logic Analyzer Object Library and click OK.

3. Copy and paste the example code below into a file (PrintLAData.py).
4. Run the example from the Command Prompt by entering the command: "python PrintLAData.py".

Example

```python
# This Python example prints all of the bus/signal’s from the first module in a 168x/9x/9xx Logic Analysis System.
#
# LAModule    -> change to the analyzer name to transfer data from (default is the first module)
# LAHostNameOrIP -> change to the hostname or IP address of the LA you're connecting to (default is 'localhost'
# LAStartRange, LAEndRange -> change to the data range to upload (default is -10 and 10 respectively)
#
# This example was tested using Python version 2.2.3 and the Python for Windows extensions build 200.

import sys
import string

# Libraries needed to interface with the Logic Analyzer COM interface.
import win32com.client
from win32com.client import constants
import pythoncom

### ==================================================================
### * Begin Subroutines *
### ------------------------------------------------------------------
##--------------------------------------------------------------------
## FUNCTION:
## PrintArrays -- prints the Logic Analyzer Data Arrays
## SYNOPSIS:
```
Using COM Automation

3

##
## ARGUMENTS:
##
arrays - array of data arrays to format and print. The
##
first array contains the bus/signal names.
##
##-------------------------------------------------------------------def PrintArrays(arrays):
NameArray = arrays[0]
TypeArray = arrays[1]
DataArrays = arrays[2:]
DataStringArrays = []
# Calculate the max width for each column and store in an array.
MaxWidthArray = []
for name in NameArray:
MaxWidthArray.append(len(name))
for column in range(len(DataArrays)):
DataStringArray = []
for dataValue in DataArrays[column]:
if TypeArray[column] == constants.AgtBusSignalProbed:
dataValueString = "%X" % dataValue
elif TypeArray[column] == constants.AgtBusSignalGenerated:
dataValueString = "%s" % dataValue
elif TypeArray[column] == constants.AgtBusSignalSampleNum:
dataValueString = "%d" % dataValue
elif TypeArray[column] == constants.AgtBusSignalTime:
dataValueString = "%E" % dataValue
DataStringArray.append(dataValueString)
dataValueStringLength = len(dataValueString)
if dataValueStringLength > MaxWidthArray[column]:
MaxWidthArray[column] = dataValueStringLength
DataStringArrays.append(DataStringArray)
# Print the header row.
print ""
for column in range(len(NameArray)):
print " " + string.center(NameArray[column], \
MaxWidthArray[column]),
print ""
for column in range(len(NameArray)):
print " " + "-" * MaxWidthArray[column],
print ""
# Print the data rows.
for row in range(len(DataStringArrays[1])):
for column in range(len(DataStringArrays)):
print " " + string.rjust(DataStringArrays[column][row], \
MaxWidthArray[column]),
print ""
### -----------------------------------------------------------------###
End of Subroutines
### ==================================================================
### ==================================================================

COM Automation Online Help

81


### * Begin Main Routine *
### ------------------------------------------------------------------

# Logic Analyzer host name or IP address.
LAHostNameOrIP = "localhost"

# Create the logic analyzer client server, and
# connect to the remote logic analyzer instrument.
print "\nConnecting to '%s'" % LAHostNameOrIP ;
try:
    LAConnect = win32com.client.Dispatch("AgtLA.Connect")
    LAInst = LAConnect.GetInstrument(LAHostNameOrIP)
except pythoncom.com_error, (hr, msg, exc, arg):
    print "The AgtLA call failed with code: %d: %s" % (hr, msg)
    if exc is None:
        print "There is no extended error information"
    else:
        wcode, source, text, helpFile, helpId, scode = exc
        print "The source of the error is", source
        print "The error message is", text
        print "More info can be found in %s (id=%d)" % (helpFile, helpId)
    sys.exit(1)

# Optionally, load a configuration file that exists on the
# logic analyzer. If the file only exists on your client PC,
# then use the LAConnect.CopyFile() method to copy the file
# onto your logic analyzer.
#
LAInst.Open("test.xml")

# Get the logic analyzer module.
#
# LAModule = LAInst.GetModuleByName("My 1691D-1")
LAModule = LAInst.Modules(0)

# Optionally, set up a trigger before running the analyzer.
#
# if LAModule.Type == "Analyzer":
#    LAAalyzerModule = \
#    win32com.client.CastTo(LAModule, "IAnalyzerModule")
#    LAAalyzerModule.SimpleTrigger("My Bus 1=hff")

# Run the analyzer and wait for the measurement to complete
# before getting data.
LAInst.Run()
LAInst.WaitComplete(10) # Time out after 10 seconds.

# Get the module's bus/signal names and types, and store
# into 'LABusSignalNameArray' and 'LABusSignalTypeArray'.
LABusSignals = LAModule.BusSignals
LABusSignalsCount = LABusSignals.Count
LABusSignalNameArray = []
LABusSignalTypeArray = []

for index in range(LABusSignalsCount):
    LABusSignalName = LABusSignals.Item(index).Name
    LABusSignalType = LABusSignals.Item(index).BusSignalType
LABusSignalNameArray.append(LABusSignalName)
LABusSignalTypeArray.append(LABusSignalType)

# Get the module's bus/signal data and store into 'LADataArrays'.
LADataArrays = []
LADataArrays.append(LABusSignalNameArray)
LADataArrays.append(LABusSignalTypeArray)
LAStartRange = -10
LAEndRange = 10

for index in range(LABusSignalsCount):
    LAData = LABusSignals.Item(index).BusSignalData
    LABusSignalType = LABusSignalTypeArray[index]
    LADataType = constants.AgtDataLong

    if LABusSignalType == constants.AgtBusSignalTime:
        LADataType = constants.AgtDataTime

    if LAData.Type == "Sample":
        SampleBusSignalData = \n            win32com.client.CastTo(LAData, "ISampleBusSignalData")
        (LADataArray, NumRows) = \n            SampleBusSignalData.GetDataBySample(LAStartRange,
                LAEndRange,
                LADataType)

        LADataArrays.append(LADataArray)

    #
    # Print the Arrays.
    #
    PrintArrays(LADataArrays)
**Using Tcl**

1. Install the Tcl software.
2. Copy and paste the example code below into a file (PrintLAData.tcl).
3. Run the example from the Command Prompt by entering the command: "tclsh.exe PrintLAData.tcl".

**Example**

```tcl
# This Tcl example prints all of the bus/signal's from the
# first module in a 168x/9x/9xx Logic Analysis System.
#
# This example was tested using ActiveState ActiveTcl 8.4.5.0
#
set usage "16900.tcl \[-e\] \[-f <hostname or IP>\] \[-c <config file>\]
Does a simple run command to the specified 16900 frame
-e : print out interface information
-f : specify the frame name on the command line
-c : specify a config file to load (optional)
",

package require cmdline; # argument processing
package require tcom; # Use the ActiveState tcom package

### ==================================================================
### * Begin Procedures *
### ------------------------------------------------------------------
##--------------------------------------------------------------------
## Procedure to explore COM interfaces
##--------------------------------------------------------------------

proc explore_tcom {handle} {
    # Explore the handle we got back....
    set ihandle [ ::tcom::info interface $handle ];
    set iname [ $ihandle name ];
    puts [ concat "Interface name: " $iname ];
    set methodlist [ $ihandle methods ];
    puts [ concat "There are " [ llength $methodlist ] \ 
        " elements in the method list" ];
    set index 0;
    while { [ llength $methodlist ] > $index } {
        set one [lindex $methodlist $index];
        set memberid [ lindex $one 0 ];
        set returntype [ lindex $one 1 ];
        set methodname [ lindex $one 2 ];
        set parmlist [ lindex $one 3 ];
        puts [ concat "name: " $methodname ", memberid: " $memberid ", \ 
            parmlist " $parmlist ];
    }
}
```
set index [ expr $index + 1 ];
}

# Explore properties of the handle
set proplist [ $ihandle properties ];

puts [ concat "\nThere are " [ llength $proplist ] \n" elements in the properties list" ];

set index 0;
while { [ llength $proplist ] > $index } {
    set one [ lindex $proplist $index ];
    set memberid [ lindex $one 0 ];
    set rwmode [ lindex $one 1 ];
    set datatype [ lindex $one 2 ];
    set propname [ lindex $one 3 ];
    set descriptions [ lindex $one 4 ];
    puts [ concat "name: " $propname ", memberid: " $memberid ", \nrwmode " $rwmode ", datatype: " $datatype ];
    set index [ expr $index + 1 ];
}
}; # explore_tcom

### ==================================================================
### End of Procedures
### ==================================================================

### * Begin Main Routine *
### ------------------------------------------------------------------

set framename "empty"; # No default frame name
set configfile "";
set exploreinterfaces 0;

# Check command line arguments
while {[ ::cmdline::getopt argv {"e" "f.arg" "c.arg"} c valvar ] > 0 } {
    switch $c 
    "c" { set configfile $valvar } \
    "e" { set exploreinterfaces 1 }\n    "f" { set framename $valvar } \
    "default" { puts $usage; exit 1; } ;
}; # while statement

# Ask for the hostname or IP address of the 16900 frame
if {($framename == "empty")} {
    puts "What is the hostname or IP of the 16900 frame? ";
    set framename [ gets stdin ];
};

puts "Connecting to '$framename';

# Open the connection to the logic analyzer
set lahandle [ ::tcom::ref createobject "AgtLA.Connect" ];
if {$lahandle == 0} {
    puts "Error opening AgtLA.Connect";
    exit 1;
}

if {$exploreinterfaces == 1 } {
    puts "*******************************
    AgtLA.Connect handle information:
    ";
    explore_tcom $lahandle;
    puts "*******************************";
}

# Attach to the frame....
set laframe [ $lahandle Instrument $framename];

if {$exploreinterfaces == 1 } {
    puts "*******************************
    Instrument handle information:
    ";
    explore_tcom $laframe;
    puts "*******************************";
}

# If they specified a config file, load it
if {$configfile ne ""} {
    set openreturn [ $laframe Open $configfile 0];
    puts [ concat "Open returned " $openreturn ];
}

# Do the run command
$laframe Run 0;   # Non repetitive run

# Wait for the run to finish
$laframe WaitComplete 10; # Wait until meas complete or 10 seconds

# Get the first analyzer's data
set analyzers [ $laframe Modules ];

if {$exploreinterfaces == 1 } {
    puts "*******************************
    Analyzers handle information:
    ";
    explore_tcom $analyzers;
    puts "*******************************";
}

# In order to pass an integer to the COM object, must
# force the internal representation to an integer using
# the following two lines:
# set intval -1;
incr intval;
set analyzer [ $analyzers Item $intval ];

if {$exploreinterfaces == 1 } {
    puts "*******************************
    Analyzer handle information:
    ";
    explore_tcom $analyzer;
    puts "*******************************";
}
analyzer handle information:
";
explore_tcom $analyzer;
puts "*******************************";
}

# Get the first analyzer's bus/signal names
set bus_signal_names [ $analyzer BusSignals ];
if {$exploreinterfaces == 1 } {
  puts "*******************************
bus_signal_names handle information:
";
explore_tcom $bus_signal_names;
puts "*******************************";
}

# Walk through the bus/signal names. Find out:
# type of data (probed, samplenum, time)
# max width of column (max of bus/signal name and printed value)
# set num_bus_signal_names [ $bus_signal_names Count ];
set index -1;
incr index; # Force to be integer 0
while {$index < $num_bus_signal_names} {
  set bus_signal_name [ $bus_signal_names Item $index ];
  if {$exploreinterfaces == 1 && $index == 0} {
    puts "*******************************
bus_signal_name handle information:
";
    explore_tcom $bus_signal_name;
    puts "*******************************";
  }
  set name [ $bus_signal_name Name ];
  set namewidth [ expr [ string length $name ] + 1 ];
  set datahandle [ $bus_signal_name BusSignalData ];
  if {$exploreinterfaces == 1 && $index == 0} {
    puts "*******************************
data handle information:
";
    explore_tcom $datahandle;
    puts "*******************************";
  }
  set bits [ $bus_signal_name BitSize ];
  # The bustype is an integer. For each type, convert to the type
  # of print-out we want
  set bustype [ $bus_signal_name BusSignalType ];
switch $bustype {
    1 { # AgtBusSignalProbed
        set datatype 6; # StringHex
        set bitwidth [expr $bits / 4 + 1];
    }
    2 { # AgtBusSignalGenerated
        set datatype 7; # DataString
        set bitwidth 30; # Arbitrary....
    }
    3 { # AgtBusSignalSampleNum
        set datatype 3; # StringDecimal
        set bitwidth [expr $bits / 10 + 1 ];
    }
    4 { # AgtBusSignalTime
        set datatype 4; # DataTime
        set bitwidth 30; # Arbitrary....
    }
}

set width $namewidth;
if {$width < $bitwidth} {set width $bitwidth};

lappend bus_signal_info $name $bits $width $datatype $datahandle;
puts "$name: $bits bits, $width width, $datatype datatype";
incr index;
}

# Print out the bus/signal names,
foreach {name bits width datatype datahandle} $bus_signal_info {
    puts -nonewline [ format "%*s " $width $name ];
}
puts "\n";

# The most efficient way to get data is to get a large
# chunk of data for the first bus/signal name, then the next, and
# so on. For the purposes of this example, just grab
# one sample at a time and go across the row....
# for {set i 0} {$i < 10} {incr i} {
    foreach {name bits width datatype datahandle} $bus_signal_info {
        set value [lindex $datahandle GetDataBySample $i $i $datatype numrows];
        puts -nonewline [ format "%*s " $width $value ];
    }
    puts "\n";
}
4

COM Automation Reference

- Objects, Methods, and Properties Quick Reference (see page 90)
- Object Hierarchy Overview (see page 102)
- Objects (Quick Reference) (see page 105)
- Methods (see page 166)
- Properties (see page 243)
### Objects, Methods, and Properties Quick Reference

<table>
<thead>
<tr>
<th>Objects</th>
<th>Methods/Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzerModule (see page 106)</td>
<td></td>
<td>A state/timing analyzer hardware measurement module.</td>
</tr>
<tr>
<td><strong>methods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetRawData (see page 200)</td>
<td></td>
<td>Given a range, returns the raw analyzer data.</td>
</tr>
<tr>
<td>GetRawTimingZoomData (see page 202)</td>
<td></td>
<td>Given a range, returns the raw analyzer timing zoom data.</td>
</tr>
<tr>
<td>RecallTriggerByName (see page 224)</td>
<td></td>
<td>Loads a named trigger from the recall buffer.</td>
</tr>
<tr>
<td>RecallTriggerByFile (see page 223)</td>
<td></td>
<td>Loads a previously saved trigger file on the instrument file system.</td>
</tr>
<tr>
<td>SimpleTrigger (see page 232)</td>
<td></td>
<td>Trigger on a simple condition with optional occurrence and storage qualification.</td>
</tr>
<tr>
<td>WaitComplete (see page 238)</td>
<td></td>
<td>Waits until the analyzer module, tool, and viewer measurements are complete.</td>
</tr>
<tr>
<td><strong>properties</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setup (see page 277)</td>
<td></td>
<td>Gets or sets the logic analyzer’s XML-format setup specification.</td>
</tr>
<tr>
<td>Trigger (see page 285)</td>
<td></td>
<td>Gets or sets the logic analyzer’s XML-format trigger specification.</td>
</tr>
<tr>
<td>BusSignal (see page 108)</td>
<td></td>
<td>A named and grouped set of pod channels.</td>
</tr>
<tr>
<td><strong>methods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsTimingZoom (see page 215)</td>
<td></td>
<td>Is this a timing zoom bus/signal?</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>(see page 245)</td>
<td>Gets the activity indicators of the bus/signal.</td>
</tr>
<tr>
<td>BitSize</td>
<td>(see page 248)</td>
<td>Gets the number of channels in the bus/signal.</td>
</tr>
<tr>
<td>BusSignalData</td>
<td>(see page 249)</td>
<td>Gets the acquisition data associated with a bus/signal.</td>
</tr>
<tr>
<td>BusSignalType</td>
<td>(see page 249)</td>
<td>Gets the type of bus/signal.</td>
</tr>
<tr>
<td>ByteSize</td>
<td>(see page 251)</td>
<td>Gets the size of the bus/signal in bytes.</td>
</tr>
<tr>
<td>Channels</td>
<td>(see page 252)</td>
<td>Gets the channels defined in the bus/signal.</td>
</tr>
<tr>
<td>CreatorName</td>
<td>(see page 259)</td>
<td>Gets the name of the module, tool, or viewer that created this bus/signal.</td>
</tr>
<tr>
<td>Name</td>
<td>(see page 269)</td>
<td>Gets or sets the name of the bus/signal.</td>
</tr>
<tr>
<td>Polarity</td>
<td>(see page 272)</td>
<td>Gets the polarity of the bus/signal.</td>
</tr>
<tr>
<td>Symbols</td>
<td>(see page 281)</td>
<td>Gets or sets the symbols associated with a bus/signal.</td>
</tr>
<tr>
<td>BusSignalData</td>
<td>(see page 108)</td>
<td>A generic bus/signal data object.</td>
</tr>
<tr>
<td>properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>(see page 285)</td>
<td>Gets the specific bus/signal data type.</td>
</tr>
<tr>
<td>BusSignalDifference</td>
<td>(see page 109)</td>
<td>Represents the different values for a particular bus/signal within a sample that has differences.</td>
</tr>
<tr>
<td>properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>(see page 269)</td>
<td>Gets the bus/signal name associated with the sample difference.</td>
</tr>
<tr>
<td>Reference</td>
<td>(see page 274)</td>
<td>Gets the reference buffer value associated with the sample difference.</td>
</tr>
<tr>
<td>Value</td>
<td>(see page 286)</td>
<td>Gets the data value associated with the sample difference.</td>
</tr>
<tr>
<td>BusSignalDifferences</td>
<td>(see page 109)</td>
<td>A collection object that contains all of the SampleDifference object’s buses/signals with differences.</td>
</tr>
<tr>
<td>properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_NewEnum</td>
<td>(see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count</td>
<td>(see page 256)</td>
<td>Gets the number of bus/signal differences in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>(see page 266)</td>
<td>Given an index into the collection, gets a BusSignalDifference (see page 109) object from the collection.</td>
</tr>
<tr>
<td>BusSignals</td>
<td>(see page 113)</td>
<td>A collection of the hardware module’s defined BusSignals.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>methods</td>
<td>Add (see page 168)</td>
<td>Adds a new bus/signal to the collection.</td>
</tr>
<tr>
<td></td>
<td>Remove (see page 224)</td>
<td>Removes a bus/signal from the collection.</td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of <strong>For Each ... Next.</strong></td>
</tr>
<tr>
<td></td>
<td>Count (see page 256)</td>
<td>Gets the number of BusSignal (see page 108) objects in the collection.</td>
</tr>
<tr>
<td></td>
<td>Item (see page 266)</td>
<td>Gets one of the BusSignal (see page 108) objects in the collection given either an index or name.</td>
</tr>
<tr>
<td>CompareWindow</td>
<td>Execute (see page 178)</td>
<td>Executes the compare using the current options.</td>
</tr>
<tr>
<td>methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>Options (see page 271)</td>
<td>Gets or sets the Compare window options.</td>
</tr>
<tr>
<td></td>
<td>SampleDifferences (see page 276)</td>
<td>Gets a collection of all the samples with differences found in the last comparison.</td>
</tr>
<tr>
<td>Connect</td>
<td>Connect (see page 118)</td>
<td>A window that compares bus/signal data.</td>
</tr>
<tr>
<td>methods</td>
<td>CopyFile (see page 173)</td>
<td>Copies a file to the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>GetRemoteInfo (see page 203)</td>
<td>Gets the logic analyzer’s remote user login and computer name.</td>
</tr>
<tr>
<td>properties</td>
<td>Instrument (see page 265)</td>
<td>Gets the logic analyzer instrument object.</td>
</tr>
<tr>
<td>ConnectSystem</td>
<td>Connect (see page 118)</td>
<td>A connection to the logic analyzer system.</td>
</tr>
<tr>
<td>methods</td>
<td>Connect (see page 172)</td>
<td>Connects to the remote logic analyzer system.</td>
</tr>
<tr>
<td></td>
<td>RecvFile (see page 224)</td>
<td>Copies a file from the remote logic analyzer system to your local system.</td>
</tr>
<tr>
<td></td>
<td>SendFile (see page 229)</td>
<td>Copies a file from your local system to the remote logic analyzer system.</td>
</tr>
<tr>
<td>FindResult</td>
<td>FindResult (see page 119)</td>
<td>Gets the results from the Find (see page 184), FindNext (see page 188), and FindPrev (see page 189) method calls.</td>
</tr>
<tr>
<td>properties</td>
<td>Found (see page 263)</td>
<td>Gets the found status.</td>
</tr>
<tr>
<td></td>
<td>OccurrencesFound (see page 270)</td>
<td>Gets the number of occurrences found.</td>
</tr>
<tr>
<td></td>
<td>SubrowFound (see page 281)</td>
<td>Gets the subrow number if found on a subrow.</td>
</tr>
<tr>
<td></td>
<td>TimeFound (see page 283)</td>
<td>Gets the time found as a double.</td>
</tr>
<tr>
<td></td>
<td>TimeFoundString (see page 284)</td>
<td>Gets the time found as a string.</td>
</tr>
<tr>
<td>Frame</td>
<td>Frame (see page 120)</td>
<td>A logic analyzer frame.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>properties</td>
<td>ComputerName (see page 255)</td>
<td>Gets the computer name of the logic analyzer frame.</td>
</tr>
<tr>
<td></td>
<td>Description (see page 260)</td>
<td>Gets a description of the logic analyzer frame.</td>
</tr>
<tr>
<td></td>
<td>IPAddress (see page 265)</td>
<td>Gets the frame’s IP address(es).</td>
</tr>
<tr>
<td></td>
<td>TargetControlPort (see page 282)</td>
<td>Gets or sets the target control port value.</td>
</tr>
<tr>
<td>Frames (see page 120)</td>
<td>A collection of logic analyzer frames connected via the multiframe connector.</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of <code>For Each ... Next</code>.</td>
</tr>
<tr>
<td></td>
<td>Count (see page 256)</td>
<td>Gets the number of frames in the collection.</td>
</tr>
<tr>
<td></td>
<td>Item (see page 266)</td>
<td>Gets one of the frames in the collection given either an index, computer name, or IP address.</td>
</tr>
<tr>
<td>Instrument (see page 121)</td>
<td>The logic analyzer instrument.</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>Close (see page 172)</td>
<td>Closes the current configuration.</td>
</tr>
<tr>
<td></td>
<td>DeleteFile (see page 173)</td>
<td>Deletes a file on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td></td>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td></td>
<td>Export (see page 179)</td>
<td>Exports data to a file on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>ExportEx (see page 180)</td>
<td>Exports data to a file on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>GetProbeByName (see page 200)</td>
<td>Given a probe name, returns its corresponding probe object.</td>
</tr>
<tr>
<td></td>
<td>GetModuleByName (see page 196)</td>
<td>Given a module name, returns its corresponding hardware module object.</td>
</tr>
<tr>
<td></td>
<td>GetToolByName (see page 205)</td>
<td>Given a tool name, returns its corresponding tool object.</td>
</tr>
<tr>
<td></td>
<td>GetWindowByName (see page 206)</td>
<td>Given a window name, returns its corresponding window object.</td>
</tr>
<tr>
<td></td>
<td>Import (see page 212)</td>
<td>Imports data from a file located on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>ImportEx (see page 213)</td>
<td>Imports data from a file located on the instrument file system into a particular module.</td>
</tr>
<tr>
<td></td>
<td>GoOffline (see page 207)</td>
<td>Disconnects the user interface from the logic analyzer frame.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>methods (cont’d)</td>
<td>GoOnline (see page 211)</td>
<td>Connects the user interface to a specific logic analyzer frame.</td>
</tr>
<tr>
<td></td>
<td>IsOnline (see page 214)</td>
<td>Tells whether the user interface is connected to a logic analyzer frame.</td>
</tr>
<tr>
<td></td>
<td>New (see page 215)</td>
<td>Creates a new instrument Overview.</td>
</tr>
<tr>
<td></td>
<td>Open (see page 216)</td>
<td>Loads a previously saved configuration file on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>PanelLock (see page 216)</td>
<td>Disables user access to the instrument front panel or remote display.</td>
</tr>
<tr>
<td></td>
<td>PanelUnlock (see page 220)</td>
<td>Re-enables user access to the instrument front panel or remote display.</td>
</tr>
<tr>
<td></td>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
<tr>
<td></td>
<td>Run (see page 227)</td>
<td>Starts running all modules.</td>
</tr>
<tr>
<td></td>
<td>Save (see page 228)</td>
<td>Saves the current configuration to a file on the instrument file system.</td>
</tr>
<tr>
<td></td>
<td>Stop (see page 236)</td>
<td>Stops all currently running modules.</td>
</tr>
<tr>
<td></td>
<td>VBADisplayHelpTopic (see page 237)</td>
<td>Displays the help page and topic for an installed VBA project.</td>
</tr>
<tr>
<td></td>
<td>VBARunMacro (see page 237)</td>
<td>Runs the specified VBA macro as if that macro was selected in the Macros dialog box.</td>
</tr>
<tr>
<td></td>
<td>VBARunRPICommand (see page 238)</td>
<td>Runs an ASCII RPI command in VBA.</td>
</tr>
<tr>
<td></td>
<td>WaitComplete (see page 238)</td>
<td>Waits until all module, tool, and viewer measurements are complete.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>properties</td>
<td>Frames (see page 263)</td>
<td>Gets a collection of logic analyzer frames connected via the multiframe connector.</td>
</tr>
<tr>
<td></td>
<td>Markers (see page 267)</td>
<td>Gets a collection of all the markers in the instrument.</td>
</tr>
<tr>
<td></td>
<td>Model (see page 268)</td>
<td>Gets the model number.</td>
</tr>
<tr>
<td></td>
<td>Modules (see page 268)</td>
<td>Gets a collection of all the hardware modules in the instrument.</td>
</tr>
<tr>
<td></td>
<td>Overview (see page 271)</td>
<td>Gets the XML-format Overview window specification.</td>
</tr>
<tr>
<td></td>
<td>PanelLocked (see page 272)</td>
<td>Indicates the front panel is locked. If locked, the message is returned.</td>
</tr>
<tr>
<td></td>
<td>Probes (see page 274)</td>
<td>Gets a collection of all currently defined probes.</td>
</tr>
<tr>
<td></td>
<td>RemoteComputerName (see page 274)</td>
<td>Gets or sets the remote computer name.</td>
</tr>
<tr>
<td></td>
<td>RemoteUserName (see page 275)</td>
<td>Gets or sets the remote user login name.</td>
</tr>
<tr>
<td></td>
<td>SelfTest (see page 277)</td>
<td>Gets the SelfTest object.</td>
</tr>
<tr>
<td></td>
<td>Status (see page 280)</td>
<td>Gets the status of all hardware modules.</td>
</tr>
<tr>
<td></td>
<td>Tools (see page 284)</td>
<td>Gets a collection of all active software tools.</td>
</tr>
<tr>
<td></td>
<td>VBAVersion (see page 287)</td>
<td>Gets the version number of VBA.</td>
</tr>
<tr>
<td></td>
<td>VBE (see page 287)</td>
<td>Gets the VBE extensibility object.</td>
</tr>
<tr>
<td></td>
<td>Version (see page 287)</td>
<td>Gets the version number of the system software.</td>
</tr>
<tr>
<td></td>
<td>Windows (see page 289)</td>
<td>Gets a collection of all active windows.</td>
</tr>
<tr>
<td>Marker (see page 123)</td>
<td>A reference point in the captured data.</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>Comments (see page 255)</td>
<td>Gets or sets the marker comments.</td>
</tr>
<tr>
<td></td>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the marker.</td>
</tr>
<tr>
<td></td>
<td>TextColor (see page 283)</td>
<td>Gets or sets the marker text color.</td>
</tr>
<tr>
<td></td>
<td>BackgroundColor (see page 247)</td>
<td>Gets or sets the marker background color.</td>
</tr>
<tr>
<td></td>
<td>Position (see page 272)</td>
<td>Gets or sets the marker position.</td>
</tr>
<tr>
<td>Markers (see page 123)</td>
<td>A collection of all the defined markers.</td>
<td></td>
</tr>
</tbody>
</table>
### Module (see page 128)

**A generic hardware module.**

<table>
<thead>
<tr>
<th>Methods/Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
<tr>
<td>WaitComplete (see page 238)</td>
<td>Waits until the module, tool, and viewer measurements are complete.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusSignals (see page 250)</td>
<td>Gets a collection of the module’s defined bus/signals.</td>
</tr>
<tr>
<td>CardModels (see page 252)</td>
<td>Gets the card model numbers.</td>
</tr>
<tr>
<td>Description (see page 260)</td>
<td>Gets a description of the module.</td>
</tr>
<tr>
<td>Frame (see page 263)</td>
<td>Gets the frame in which the module resides.</td>
</tr>
<tr>
<td>Model (see page 268)</td>
<td>Gets the model number.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the module.</td>
</tr>
<tr>
<td>RunningStatus (see page 275)</td>
<td>Gets the detailed running status of the module.</td>
</tr>
<tr>
<td>Slot (see page 278)</td>
<td>Gets the module’s slot location in the frame.</td>
</tr>
<tr>
<td>Status (see page 280)</td>
<td>Gets the status of the module.</td>
</tr>
<tr>
<td>StatusMsg (see page 281)</td>
<td>Gets the formatted status message.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the specific module type.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Modules (see page 129)</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td></td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td></td>
</tr>
<tr>
<td>PattgenModule (see page 129)</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>GetLine (see page 194)</td>
</tr>
<tr>
<td></td>
<td>GetLineLabel (see page 196)</td>
</tr>
<tr>
<td></td>
<td>InsertLine (see page 214)</td>
</tr>
<tr>
<td></td>
<td>RemoveLine (see page 226)</td>
</tr>
<tr>
<td></td>
<td>Reset (see page 226)</td>
</tr>
<tr>
<td></td>
<td>Resume (see page 227)</td>
</tr>
<tr>
<td></td>
<td>Run (see page 228)</td>
</tr>
<tr>
<td></td>
<td>SetLine (see page 230)</td>
</tr>
<tr>
<td></td>
<td>SetLineLabel (see page 230)</td>
</tr>
<tr>
<td></td>
<td>Step (see page 235)</td>
</tr>
<tr>
<td></td>
<td>Stop (see page 236)</td>
</tr>
<tr>
<td>properties</td>
<td>NumLines (see page 270)</td>
</tr>
<tr>
<td>Probe (see page 135)</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>DoAction (see page 174)</td>
</tr>
<tr>
<td></td>
<td>DoCommands (see page 174)</td>
</tr>
<tr>
<td></td>
<td>QueryCommand (see page 220)</td>
</tr>
<tr>
<td>properties</td>
<td>Name (see page 269)</td>
</tr>
<tr>
<td></td>
<td>Type (see page 285)</td>
</tr>
<tr>
<td>Probes (see page 135)</td>
<td></td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
</tr>
<tr>
<td></td>
<td>Count (see page 256)</td>
</tr>
<tr>
<td></td>
<td>Item (see page 266)</td>
</tr>
<tr>
<td>SampleBusSignalData</td>
<td>(see page 138)</td>
</tr>
<tr>
<td>methods</td>
<td>GetDataBySample (see page 189)</td>
</tr>
<tr>
<td></td>
<td>GetDataByTime (see page 192)</td>
</tr>
<tr>
<td></td>
<td>GetNumSamples (see page 199)</td>
</tr>
<tr>
<td></td>
<td>GetSampleNumByTime (see page 204)</td>
</tr>
<tr>
<td></td>
<td>GetTime (see page 204)</td>
</tr>
<tr>
<td>properties</td>
<td>DataType (see page 260)</td>
</tr>
<tr>
<td></td>
<td>EndSample (see page 261)</td>
</tr>
<tr>
<td></td>
<td>EndTime (see page 262)</td>
</tr>
<tr>
<td></td>
<td>StartSample (see page 279)</td>
</tr>
<tr>
<td></td>
<td>StartTime (see page 279)</td>
</tr>
<tr>
<td></td>
<td>SampleDifference (see page 149)</td>
</tr>
<tr>
<td>properties</td>
<td>BusSignalDifferences (see page 250)</td>
</tr>
<tr>
<td></td>
<td>SampleNum (see page 277)</td>
</tr>
<tr>
<td></td>
<td>SampleDifferences (see page 149)</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of bus/signal differences in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Given an index into the collection, gets a SampleDifference (see page 149) object from the collection.</td>
</tr>
<tr>
<td>SelfTest (see page 149)</td>
<td>Object for running instrument self-tests.</td>
</tr>
<tr>
<td>methods</td>
<td>TestAll (see page 237)</td>
</tr>
<tr>
<td>Tool (see page 152)</td>
<td>A generic instrument software tool.</td>
</tr>
<tr>
<td>methods</td>
<td>DoAction (see page 174)</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
<tr>
<td>properties</td>
<td>BusSignals (see page 250)</td>
</tr>
<tr>
<td>Name (see page 289)</td>
<td>Gets or sets the name of the tool.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the specific tool type.</td>
</tr>
<tr>
<td>Tools (see page 153)</td>
<td>A collection of all of the instrument’s software tools.</td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of tools in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the tools in the collection given either an index or name.</td>
</tr>
<tr>
<td>VbaViewChart (see page 156)</td>
<td>A chart for the VbaViewWindow.</td>
</tr>
<tr>
<td>methods</td>
<td>Draw (see page 178)</td>
</tr>
<tr>
<td>properties</td>
<td>Axis (see page 246)</td>
</tr>
<tr>
<td>ChartType (see page 253)</td>
<td>Gets or sets the chart type.</td>
</tr>
<tr>
<td>Data (see page 260)</td>
<td>Gets the chart data.</td>
</tr>
<tr>
<td>HasLegend (see page 264)</td>
<td>Gets or sets if the legend is visible.</td>
</tr>
<tr>
<td>HasTitle (see page 264)</td>
<td>Gets or sets if the title is visible.</td>
</tr>
<tr>
<td>Legend (see page 267)</td>
<td>Gets the chart legend.</td>
</tr>
<tr>
<td>Title (see page 284)</td>
<td>Gets the title of the chart.</td>
</tr>
<tr>
<td>VbaViewChartAxis (see page 157)</td>
<td>The axis of a VbaViewChart.</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
</tr>
<tr>
<td>properties</td>
<td>AxisBase (see page 246)</td>
</tr>
<tr>
<td></td>
<td>BitSize (see page 248)</td>
</tr>
<tr>
<td></td>
<td>HasTitle (see page 264)</td>
</tr>
<tr>
<td></td>
<td>Title (see page 284)</td>
</tr>
<tr>
<td>VbaViewChartData (see page 157)</td>
<td>The data of a VbaViewChart.</td>
</tr>
<tr>
<td>methods</td>
<td>AddPointArrays (see page 170)</td>
</tr>
<tr>
<td></td>
<td>Clear (see page 171)</td>
</tr>
<tr>
<td></td>
<td>GetGroupCaption (see page 193)</td>
</tr>
<tr>
<td></td>
<td>GetValueCaption (see page 206)</td>
</tr>
<tr>
<td></td>
<td>SetGroupCaption (see page 229)</td>
</tr>
<tr>
<td></td>
<td>SetValue (see page 231)</td>
</tr>
<tr>
<td></td>
<td>SetValueArray (see page 231)</td>
</tr>
<tr>
<td></td>
<td>SetValueCaption (see page 232)</td>
</tr>
<tr>
<td>VbaViewChartFont (see page 158)</td>
<td>The font of a VbaViewChartTitle.</td>
</tr>
<tr>
<td>properties</td>
<td>Bold (see page 248)</td>
</tr>
<tr>
<td></td>
<td>Color (see page 254)</td>
</tr>
<tr>
<td></td>
<td>FaceName (see page 262)</td>
</tr>
<tr>
<td></td>
<td>Size (see page 278)</td>
</tr>
<tr>
<td>VbaViewChartLegend (see page 158)</td>
<td>The legend of a VbaViewChart.</td>
</tr>
<tr>
<td>properties</td>
<td>Position (see page 273)</td>
</tr>
<tr>
<td>VbaViewChartTitle (see page 159)</td>
<td>The title of a VbaViewChart.</td>
</tr>
<tr>
<td>properties</td>
<td>Caption (see page 251)</td>
</tr>
<tr>
<td></td>
<td>Font (see page 262)</td>
</tr>
<tr>
<td>VbaViewWebBrowser (see page 159)</td>
<td>A web browser for the VbaViewWindow.</td>
</tr>
<tr>
<td>methods</td>
<td>Clear (see page 171)</td>
</tr>
<tr>
<td>properties</td>
<td>WebBrowser (see page 288)</td>
</tr>
<tr>
<td>Objects</td>
<td>Methods/Properties</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>VbaViewWindow (see page 160)</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>ClearOutput (see page 172)</td>
</tr>
<tr>
<td></td>
<td>WriteOutput (see page 242)</td>
</tr>
<tr>
<td>properties</td>
<td>Chart (see page 253)</td>
</tr>
<tr>
<td></td>
<td>WebBrowser (see page 288)</td>
</tr>
<tr>
<td>Window (see page 160)</td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td>DoAction (see page 174)</td>
</tr>
<tr>
<td></td>
<td>DoCommands (see page 174)</td>
</tr>
<tr>
<td></td>
<td>Find (see page 184)</td>
</tr>
<tr>
<td></td>
<td>FindNext (see page 188)</td>
</tr>
<tr>
<td></td>
<td>FindPrev (see page 189)</td>
</tr>
<tr>
<td></td>
<td>GoToPosition (see page 212)</td>
</tr>
<tr>
<td></td>
<td>QueryCommand (see page 220)</td>
</tr>
<tr>
<td>properties</td>
<td>BusSignals (see page 250)</td>
</tr>
<tr>
<td></td>
<td>Name (see page 269)</td>
</tr>
<tr>
<td></td>
<td>Type (see page 285)</td>
</tr>
<tr>
<td>Windows (see page 161)</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>_NewEnum (see page 289)</td>
</tr>
<tr>
<td></td>
<td>Count (see page 256)</td>
</tr>
<tr>
<td></td>
<td>Item (see page 266)</td>
</tr>
</tbody>
</table>

**See Also**  
- Logic Analyzer Object Hierarchy Overview (see page 102)
Object Hierarchy Overview

The Instrument object represents the logic analysis system. From the Instrument object, you can directly access objects by using the Instrument objects properties and methods, or, you can indirectly access objects through other objects obtained by these properties and methods.

The Instrument object contains collections of: Modules (see page 129) (that represent the hardware installed in the instrument), Probes (see page 135) (that organize probes connected to a DUT), Tools (see page 153) (that filter or decode captured data), and Windows (see page 161) (that display captured data). When the instrument is initially powered up, the Probes and Tools collections are empty and are not available until they are created in the user interface or restored by opening configuration files.

Module (see page 128), Tool (see page 152), and Window (see page 160) objects return data through a BusSignalData (see page 108) object, which returns information about:

- Directly acquired data when obtained from a Module (see page 128) object.
- Created and/or manipulated data when obtained from a Tool (see page 152) object.
- Displayed data when obtained from a Window (see page 160) object.

The following tree illustrates the hierarchy of the Instrument object:

- Connect (see page 118) — Not used in integrated VBA environment.
- Instrument (see page 121) — Not used in integrated VBA environment.
  - Frames (see page 120)
    - Frame (see page 120)
  - Markers (see page 123)
    - Marker (see page 123)
  - SelfTest (see page 149)
  - Modules (see page 129) — Collection of all modules in the system.
    - Module (see page 128) — Generic module object.
      - BusSignals (see page 113)
        - BusSignal (see page 108)
          - BusSignalData (see page 108)
          - SampleBusSignalData (see page 138)
        - AnalyzerModule (see page 106) — Logic analyzer specific object.
• PattgenModule (see page 129) — Pattern generator specific object.
• Probes (see page 135)
  • Probe (see page 135)
• Tools (see page 153)
  • Tool (see page 152)
    • BusSignals (see page 113)
      • BusSignal (see page 108)
        • BusSignalData (see page 108) — Generic bus/signal data object.
        • SampleBusSignalData (see page 138) — Sample bus/signal data specific object.
• Windows (see page 161)
  • Window (see page 160) — Generic window object.
    • BusSignals (see page 113)
      • BusSignal (see page 108)
        • BusSignalData (see page 108)
        • SampleBusSignalData (see page 138)
    • FindResult (see page 119)
    • CompareWindow (see page 118) — Compare window specific object.
      • SampleDifferences (see page 149)
        • SampleDifference (see page 149)
        • BusSignalDifferences (see page 109)
        • BusSignalDifference (see page 109)
    • VbaViewWindow (see page 160) — VbaView window specific object.
      • VbaViewChart (see page 156)
        • VbaViewChartAxis (see page 157)
        • VbaViewChartData (see page 157)
        • VbaViewChartLegend (see page 158)
        • VbaViewChartTitle (see page 159)
        • VbaViewChartFont (see page 158)
      • VbaViewWebBrowser (see page 159)
• ConnectSystem (see page 119)
There are generic and specific objects. For example:

- The Module (see page 128) object is generic; it contains the methods and properties that are common to both the AnalyzerModule (see page 106) object and the PattgenModule (see page 129) object.
- The AnalyzerModule (see page 106) object contains logic analyzer specific properties and methods (such as the GetDataBySample (see page 189) method), but it also has access to all of the generic properties and methods in the Module (see page 128) object.
- The PattgenModule (see page 129) object contains pattern generator specific properties and methods (such as the InsertLine (see page 214) method), but it also has access to all of the generic properties and methods in the Module (see page 128) object.

If you know what type of object you have, use the specific objects (like AnalyzerModule (see page 106) and PattgenModule (see page 129)).

If you don't know what type of object you have, use the generic object (like Module (see page 128)); then, depending on the type, you can use the more specific objects. For example:

```vba
Dim myModule As AgtLA.Module
Set myModule = AgtLA.Modules(0) ' Start generic (integrated VBA env).

Dim myAnalyzerModule As AgtLA.AnalyzerModule
If (myModule.Type = "Analyzer") Then ' Once you know the type, 
   Set myAnalyzerModule = myModule ' use the more specific object 
End If ' (coerce).

Dim myPattgenModule As AgtLA.PattgenModule
If (myModule.Type = "Pattgen") Then ' Once you know the type, 
   Set myPattgenModule = myModule ' use the more specific object 
End If ' (coerce).
```

**See Also**
- Object Quick Reference (see page 105)
Object Quick Reference

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnalyzerModule (see page 106)</td>
<td>A state/timing analyzer hardware measurement module.</td>
</tr>
<tr>
<td>BusSignal (see page 108)</td>
<td>A named and grouped set of pod channels.</td>
</tr>
<tr>
<td>BusSignalData (see page 108)</td>
<td>A generic bus/signal data object.</td>
</tr>
<tr>
<td>BusSignalDifference (see page 109)</td>
<td>Represents the different values for a particular bus/signal within a</td>
</tr>
<tr>
<td></td>
<td>sample that has differences.</td>
</tr>
<tr>
<td>BusSignalDifferences (see page 109)</td>
<td>A collection object that contains all of the SampleDifference</td>
</tr>
<tr>
<td></td>
<td>object’s buses/signals with differences.</td>
</tr>
<tr>
<td>BusSignals (see page 113)</td>
<td>A collection of the hardware module’s defined BusSignals.</td>
</tr>
<tr>
<td>CompareWindow (see page 118)</td>
<td>A window that compares bus/signal data.</td>
</tr>
<tr>
<td>Connect (see page 118)</td>
<td>A connection to the logic analyzer instrument.</td>
</tr>
<tr>
<td>ConnectSystem (see page 119)</td>
<td>A connection to the logic analyzer system.</td>
</tr>
<tr>
<td>FindResult (see page 119)</td>
<td>Gets the results from the Find (see page 184), FindNext (see page 188), and</td>
</tr>
<tr>
<td></td>
<td>FindPrev (see page 189) method calls.</td>
</tr>
<tr>
<td>Frame (see page 120)</td>
<td>A logic analyzer frame.</td>
</tr>
<tr>
<td>Frames (see page 120)</td>
<td>A collection of logic analyzer frames connected via the multiframe connector.</td>
</tr>
<tr>
<td>Instrument (see page 121)</td>
<td>The logic analyzer instrument.</td>
</tr>
<tr>
<td>Marker (see page 123)</td>
<td>A reference point in the captured data.</td>
</tr>
<tr>
<td>Markers (see page 123)</td>
<td>A collection of all the defined markers.</td>
</tr>
<tr>
<td>Module (see page 128)</td>
<td>A generic hardware module.</td>
</tr>
<tr>
<td>Modules (see page 129)</td>
<td>A collection of all the hardware modules installed in the instrument.</td>
</tr>
<tr>
<td>PattgenModule (see page 129)</td>
<td>A pattern generator hardware module.</td>
</tr>
<tr>
<td>Probe (see page 135)</td>
<td>A generic probe.</td>
</tr>
<tr>
<td>Probes (see page 135)</td>
<td>A collection of all active probes.</td>
</tr>
<tr>
<td>SampleBusSignalData (see page 138)</td>
<td>The data for a specific bus/signal captured in the state or timing</td>
</tr>
<tr>
<td></td>
<td>sampling modes.</td>
</tr>
</tbody>
</table>
### AnalyzerModule Object

The **AnalyzerModule** object represents a state/timing analyzer hardware measurement module.

#### See Also
- Logic Analyzer Object Hierarchy Overview (see page 102)

#### Description

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SampleDifference (see page 149)</td>
<td>Represents a sample containing a CompareWindow difference.</td>
</tr>
<tr>
<td>SampleDifferences (see page 149)</td>
<td>A collection object of all the sample differences in the CompareWindow object.</td>
</tr>
<tr>
<td>SelfTest (see page 149)</td>
<td>Provides methods for running instrument self-tests.</td>
</tr>
<tr>
<td>Tool (see page 152)</td>
<td>A generic instrument software tool.</td>
</tr>
<tr>
<td>Tools (see page 153)</td>
<td>A collection of all of the instrument’s software tools.</td>
</tr>
<tr>
<td>VbaViewChart (see page 156)</td>
<td>A chart for the VbaViewWindow.</td>
</tr>
<tr>
<td>VbaViewChartAxis (see page 157)</td>
<td>The axis of a VbaViewChart.</td>
</tr>
<tr>
<td>VbaViewChartData (see page 157)</td>
<td>The data of a VbaViewChart.</td>
</tr>
<tr>
<td>VbaViewChartFont (see page 158)</td>
<td>The font properties of a VbaViewChartTitle.</td>
</tr>
<tr>
<td>VbaViewChartLegend (see page 158)</td>
<td>The legend of a VbaViewChart.</td>
</tr>
<tr>
<td>VbaViewChartTitle (see page 159)</td>
<td>The title of a VbaViewChart.</td>
</tr>
<tr>
<td>VbaViewWebBrowser (see page 159)</td>
<td>A web browser for the VbaViewWindow.</td>
</tr>
<tr>
<td>VbaViewWindow (see page 160)</td>
<td>A window for Visual Basic for Applications (VBA) program output.</td>
</tr>
<tr>
<td>Window (see page 160)</td>
<td>A generic instrument display window.</td>
</tr>
<tr>
<td>Windows (see page 161)</td>
<td>A collection of all of the instrument’s display windows.</td>
</tr>
</tbody>
</table>

#### To Access

```vba
Dim variable As AgtLA.AnalyzerModule
Set variable = Module (see page 128)
```
Since this object is derived from the Module (see page 128) object, a Set must be used to get to these specific methods and properties. For example:

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the online help),
' the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst"
' with "AgtLA" to access the global Instrument object in VBA.
'
' Get the AnalyzerModule specific object.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.Modules(0)
MsgBox "Trigger: " + myAnalyzer.Trigger

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRawData (see page 200)</td>
<td>Given a range, returns the raw analyzer data.</td>
</tr>
<tr>
<td>GetRawTimingZoomData (see page 202)</td>
<td>Given a range, returns the raw analyzer timing zoom data.</td>
</tr>
<tr>
<td>RecallTriggerByName (see page 224)</td>
<td>Loads a named trigger from the recall buffer.</td>
</tr>
<tr>
<td>RecallTriggerByFile (see page 223)</td>
<td>Loads a previously saved trigger file on the instrument file system.</td>
</tr>
<tr>
<td>SimpleTrigger (see page 232)</td>
<td>Trigger on a simple condition with optional occurrence and storage qualification.</td>
</tr>
<tr>
<td>WaitComplete (see page 238)</td>
<td>Waits until the analyzer module, tool, and viewer measurements are complete.</td>
</tr>
</tbody>
</table>

(Also Includes Module (see page 128) object methods)

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup (see page 277)</td>
<td>Gets or sets the logic analyzer’s XML-format setup specification.</td>
</tr>
<tr>
<td>Trigger (see page 285)</td>
<td>Gets or sets the logic analyzer’s XML-format trigger specification.</td>
</tr>
</tbody>
</table>

(Also Includes Module (see page 128) object properties)
BusSignal Object

To Access

- BusSignals.Item (see page 266) IndexOrName
- BusSignals IndexOrName

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsTimingZoom (see page 215)</td>
<td>Is this a timing zoom bus/signal?</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity (see page 245)</td>
<td>Gets the activity indicators of the bus/signal.</td>
</tr>
<tr>
<td>BitSize (see page 248)</td>
<td>Gets the number of channels in the bus/signal.</td>
</tr>
<tr>
<td>BusSignalData (see page 249)</td>
<td>Gets the acquisition data associated with a bus/signal.</td>
</tr>
<tr>
<td>BusSignalType (see page 249)</td>
<td>Gets the type of bus/signal.</td>
</tr>
<tr>
<td>ByteSize (see page 251)</td>
<td>Gets the size of the bus/signal in bytes.</td>
</tr>
<tr>
<td>Channels (see page 252)</td>
<td>Gets the channels defined in the bus/signal.</td>
</tr>
<tr>
<td>CreatorName (see page 259)</td>
<td>Gets the name of the module, tool, or viewer that created this bus/signal.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the bus/signal.</td>
</tr>
<tr>
<td>Polarity (see page 272)</td>
<td>Gets the polarity of the bus/signal.</td>
</tr>
<tr>
<td>Symbols (see page 281)</td>
<td>Gets or sets the symbols associated with a bus/signal.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the specific bus/signal data type.</td>
</tr>
</tbody>
</table>

See Also

- BusSignals (see page 113) object

BusSignalData Object

To Access

- BusSignal.BusSignalData (see page 249)

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Gets the specific bus/signal data type.</td>
</tr>
</tbody>
</table>
See Also  • BusSignalData (see page 249) property

BusSignalDifference Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]

To Access  • BusSignalDifferences.Item (see page 266) IndexOrName
          • BusSignalDifferences IndexOrName

Methods  There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name (see page 269)</td>
<td>Gets the bus/signal name associated with the sample difference.</td>
</tr>
<tr>
<td>Reference (see page 274)</td>
<td>Gets the reference buffer value associated with the sample difference.</td>
</tr>
<tr>
<td>Value (see page 286)</td>
<td>Gets the data value associated with the sample difference.</td>
</tr>
</tbody>
</table>

See Also  • BusSignalDifferences (see page 109) object

BusSignalDifferences Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]

To Access  • SampleDifference.BusSignalDifferences (see page 250)

Methods  There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of bus/signal differences in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Given an index into the collection, gets a BusSignalDifference (see page 109) object from the collection.</td>
</tr>
</tbody>
</table>

See Also  • BusSignalDifferences (see page 250) property

BusSignalDifferences Example

Visual Basic  ' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it ' to access the Instrument object. In this example, "myInst"
' represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the
online help),
' the Instrument object is already created and is globally
' accessible using "AgtLA". In this example, substitute "myInst"
' with "AgtLA" to access the global Instrument object in VBA.

' Load the configuration file.
myInst.Open ("c:\LA\Configs\mpc860_demo_compare.ala")

' Run the measurement, wait for it to complete.
myInst.Run
myInst.WaitComplete (20)

' Get the CompareWindow object.
Dim myCompare As AgtLA.CompareWindow
Set myCompare = myInst.GetWindowByName("Compare-1")

' Set the CompareWindow options.
Dim myOptions As String
myOptions = "<Options ReferenceOffset='0' Range='M1..M2' " + _
   "MaxDifferences='0'/">
myCompare.Options = myOptions

' Display the CompareWindow options.
myOptions = myCompare.Options
MsgBox "CompareWindow Options: " + myOptions

' Execute the compare.
myCompare.Execute

' Display the bus/signal differences between M1 and M2.
Dim mySampleDiff As AgtLA.SampleDifference
Dim myBusSignalDiff As AgtLA.BusSignalDifference
Dim myString As String

For Each mySampleDiff In myCompare.SampleDifferences
    myString = myString + vbNewLine + "Sample: " + _
        Str(mySampleDiff.SampleNum)
    For Each myBusSignalDiff In mySampleDiff.BusSignalDifferences
        ' Add the bus signal difference information to the string.
        myString = myString + ", Bus/signal: " + myBusSignalDiff.Name
        myString = myString + ", Value: " + myBusSignalDiff.Value
        myString = myString + ", Ref: " + myBusSignalDiff.Reference
    Next
Next
MsgBox "BusSignal difference values: " + vbNewLine + myString

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to display bus/signal
// differences in the Compare window.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
//
```c
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

//////////////////////////////////////////////////////////////////////
// main() entry point.
//////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[]) {
    printf("*** Main()
");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr)) {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            printf("Connecting to instrument '%s'", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\\LA\\Configs\\compare.ala";
            printf("Loading the config file '%s'", (char*) configFile);
            pInst->Open(configFile, FALSE, ",", TRUE);

            // Run the measurement, wait for it to complete.
            pInst->Run(FALSE);
        }
    }
    return 0;
}
```
pInst->WaitComplete(20);

// Get the CompareWindow object.
AgtLA::ICompareWindowPtr pCompareWindow =
    pInst->GetWindowByName("Compare-1");

// Set the CompareWindow options.
_bstr_t myCompareOptions = "<Options ReferenceOffset='0' \ 
    Range='M1..M2' MaxDifferences='0'/>";
pCompareWindow->PutOptions(myCompareOptions);

// Display the CompareWindow options.
myCompareOptions = pCompareWindow->GetOptions();
printf("CompareWindow options: '%s'\n", 
    (char*) myCompareOptions);

// Execute the compare.
pCompareWindow->Execute();

// Display the bus/signal differences between M1 and M2.
AgtLA::ISampleDifferencesPtr pSampleDifferences =
    pCompareWindow->GetSampleDifferences();
for (long i = 0; i < pSampleDifferences->GetCount(); i++)
{
    AgtLA::ISampleDifferencePtr pSampleDifference =
        pSampleDifferences->GetItem(i);
    printf("Sample: '%d'\n", pSampleDifference->GetSampleNum());

    AgtLA::IBusSignalDifferencesPtr pBusSignalDifferences =
        pSampleDifference->GetBusSignalDifferences();
    for (long j = 0; j < pBusSignalDifferences->GetCount(); j++)
    {
        // Print the bus signal difference information.
        AgtLA::IBusSignalDifferencePtr pBusSignalDifference =
            pBusSignalDifferences->GetItem(j);
        printf(" Bus/signal: '%s'\n", 
            (char*) pBusSignalDifference->GetName());
        printf(" Value: '%s'\n", 
            (char*) pBusSignalDifference->GetValue());
        printf(" Ref: '%s'\n", 
            (char*) pBusSignalDifference->GetReference());
    }
}
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();

else
{
    printf("CoInitialize failed\n");
}

return 0;
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();
    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }

    printf(" Error Message = %s\n", (char*) errorStr);
}

**BusSignals Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 114) ]

To Access
- Module.BusSignals (see page 250)
- Tool.BusSignals (see page 250)
- Window.BusSignals (see page 250)

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (see page 168)</td>
<td>Adds a new bus/signal to the collection.</td>
</tr>
<tr>
<td>Remove (see page 224)</td>
<td>Removes a bus/signal from the collection.</td>
</tr>
</tbody>
</table>
**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of BusSignal (see page 108) objects in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the BusSignal (see page 108) objects in the collection given either an index or name.</td>
</tr>
</tbody>
</table>

**Remarks**

<table>
<thead>
<tr>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The &quot;Time&quot; data is no longer returned as a bus/signal (see What’s Changed (see page 293)).</td>
</tr>
</tbody>
</table>

**See Also**

- BusSignals (see page 250) property

**BusSignals Example**

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Display bus/signal information.
Dim myBusSignals As AgtLA.BusSignals
Set myBusSignals = myInst.GetModuleByName("My 1690A-1").BusSignals

Dim myBusSignal As AgtLA.BusSignal
Dim myString As String

For Each myBusSignal In myBusSignals
    myString = myString + "Name: " + myBusSignal.Name + vbNewLine
    myString = myString + "BitSize=" + Str(myBusSignal.BitSize) + vbNewLine
    myString = myString + ", ByteSize=" + Str(myBusSignal.ByteSize) + vbNewLine
    myString = myString + ", Type=" + myBusSignal.BusSignalType + vbNewLine
    Select Case myBusSignal.BusSignalType
        Case AgtBusSignalSampleNum
            myString = myString + "AgtBusSignalSampleNum" + vbNewLine
        Case AgtBusSignalTime
            myString = myString + "AgtBusSignalTime" + vbNewLine
        Case AgtBusSignalGenerated
            myString = myString + "AgtBusSignalGenerated" + vbNewLine
        Case AgtBusSignalProbed
            myString = myString + "AgtBusSignalProbed" + vbNewLine
    End Select
Next

' The "Time" data is no longer returned as a bus/signal (see What’s Changed (see page 293)).
myString = myString + " Channels=" + 
    myBusSignal.Channels + vbCrLf
myString = myString + " Polarity=" + 
    myBusSignal.Polarity + vbCrLf
myString = myString + " Activity=" + 
    myBusSignal.Activity + vbCrLf
Case Else
    myString = myString + "Unknown" + vbCrLf
End Select
Next
MsgBox "Bus/signal information: " + myString

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to display bus/signal
// information.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

//////////////////////////////////////////////////////////////////////
// main() entry point.
//////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
if (SUCCEEDED(hr))
{
    try {  // Catch any unexpected run-time errors.
        _bstr_t hostname = "mtx33";  // TODO, use your logic
        printf(\"Connecting to instrument \"%s\"\n", (char*) hostname);

        // Create the connect object and get the instrument object.
        AgtLA::IConnectPtr pConnect = 
            AgtLA::IObjectPtr(__uuidof(AgtLA::Connect));
        AgtLA::IInstrumentPtr pInst = 
            pConnect->GetInstrument(hostname);

        // Load the configuration file.
        _bstr_t configFile = "C:\LA\Configs\config.ala";
        printf(\"Loading the config file \"%s\"\n", (char*) configFile);
        pInst->Open(configFile, FALSE, "", TRUE);

        // Get module whose bus/signal information will be displayed.
        _bstr_t moduleName = "MPC860 Demo Board";
        AgtLA::IModulePtr pModule = pInst->GetModuleByName(moduleName);

        // For each bus/signal, display a range of data.
        AgtLA::IBusSignalsPtr pBusSignals = pModule->GetBusSignals();
        for (long i = 0; i < pBusSignals->GetCount(); i++)
        {
            // Get the data for the bus/signal.
            AgtLA::IBusSignalPtr pBusSignal = 
                pBusSignals->GetItem(i);
            _bstr_t busSignal = pBusSignal->GetName();
            printf(\" Name: \"%s\"\n", (char*) busSignal);
            printf(\" BitSize: \"%d\"\n", pBusSignal->GetBitSize());
            printf(\" ByteSize: \"%d\"\n", pBusSignal->GetByteSize());

            switch(pBusSignal->GetBusSignalType())
            {
            case AgtLA::AgtBusSignalProbed:
                printf(\" Type: \"AgtBusSignalProbed\"\n");
                break;
            case AgtLA::AgtBusSignalGenerated:
                printf(\" Type: \"AgtBusSignalGenerated\"\n");
                break;
            case AgtLA::AgtBusSignalSampleNum:
                printf(\" Type: \"AgtBusSignalSampleNum\"\n");
                break;
            case AgtLA::AgtBusSignalTime:
                break;
            }
printf(" Type: 'AgtBusSignalTime'\n");
}

default:
  break;
}
}
catch (_com_error& e) {
  DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
} else {
  printf("CoInitialize failed\n");
}
return 0;
}

// Displays the last error -- used to show the last exception information.
void DisplayError(_com_error& error)
{
  printf("*** DisplayError()\n");
  printf("Fatal Unexpected Error:\n");
  printf(" Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();

  if (desc.length() == 0) {
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
      errorStr[nLen - 2] = '\0';
  } else {
    strcpy(errorStr, desc);
  }
printf(" Error Message = %s\n", (char*) errorStr);

**CompareWindow Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]

To Access

\[
\begin{align*}
\text{Dim variable As AgtLA.CompareWindow} \\
\text{Set variable = Window (see page 160)}
\end{align*}
\]

Description

The **CompareWindow** object represents an instrument window that compares bus/signal data.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute (see page 178)</td>
<td>Executes the compare using the current options.</td>
</tr>
</tbody>
</table>

(Also Includes Window (see page 160) methods)

### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options (see page 271)</td>
<td>Gets or sets the Compare window options.</td>
</tr>
<tr>
<td>SampleDifferences (see page 276)</td>
<td>Gets a collection of all the samples with differences found in the last comparison.</td>
</tr>
</tbody>
</table>

(Also Includes Window (see page 160) properties)

**Connect Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

To Access

\[
\begin{align*}
\text{Dim variable As AgtLA.Connect} \\
\text{Set variable = CreateObject("AgtLA.Connect")}
\end{align*}
\]

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyFile (see page 173)</td>
<td>Copies a file to the instrument file system.</td>
</tr>
<tr>
<td>GetRemoteInfo (see page 203)</td>
<td>Gets the logic analyzer's remote user login and computer name.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument (see page 265)</td>
<td>Gets the logic analyzer instrument object.</td>
</tr>
</tbody>
</table>

**ConnectSystem Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

To Access

- Dim variable As AgtLA.ConnectSystem
- Set variable = CreateObject("AgtLA.ConnectSystem")

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect (see page 172)</td>
<td>Connects to the remote logic analyzer system.</td>
</tr>
<tr>
<td>RecvFile (see page 224)</td>
<td>Copies a file from the remote logic analyzer system to your local system.</td>
</tr>
<tr>
<td>SendFile (see page 229)</td>
<td>Copies a file from your local system to the remote logic analyzer system.</td>
</tr>
</tbody>
</table>

Properties

There are no properties.

**FindResult Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 185) ]

To Access

- Window.Find(Event) (see page 184)
- Window.FindNext() (see page 188)
- Window.FindPrev() (see page 189)

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found (see page 263)</td>
<td>Gets the found status.</td>
</tr>
<tr>
<td>OccurrencesFound (see page 270)</td>
<td>Gets the number of occurrences found.</td>
</tr>
<tr>
<td>SubrowFound (see page 281)</td>
<td>Gets the subrow number if found on a subrow.</td>
</tr>
<tr>
<td>TimeFound (see page 283)</td>
<td>Gets the time found as a double.</td>
</tr>
<tr>
<td>TimeFoundString (see page 284)</td>
<td>Gets the time found as a string.</td>
</tr>
</tbody>
</table>
See Also
- Find (see page 184) method
- FindNext (see page 188) method
- FindPrev (see page 189) method
- GetDataByTime (see page 192) method

Frame Object

To Access
- Frames.Item (see page 266) IndexOrName
- Frames IndexOrName

Methods
There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComputerName (see page 255)</td>
<td>Gets the computer name of the logic analyzer frame.</td>
</tr>
<tr>
<td>Description (see page 260)</td>
<td>Gets a description of the logic analyzer frame.</td>
</tr>
<tr>
<td>IPAddress (see page 265)</td>
<td>Gets the frame’s IP address(es).</td>
</tr>
<tr>
<td>TargetControlPort (see page 282)</td>
<td>Gets or sets the target control port value.</td>
</tr>
</tbody>
</table>

See Also
- Frames (see page 120) object

Frames Object

To Access
- Instrument.Frames (see page 263)

Methods
There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of frames in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the frames in the collection given either an index, computer name, or IP address.</td>
</tr>
</tbody>
</table>

See Also
- Frames (see page 263) property
Instrument Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

To Access

- When using Visual Basic outside of the *Agilent Logic Analyzer* application:

  Dim variable As AgtLA.Instrument
  Set variable = Connect.Instrument (see page 265) HostNameOrIpAddress

- When "using the Advanced Customization Environment (ACE)" (in the online help):

  AgtLA

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close (see page 172)</td>
<td>Closes the current configuration.</td>
</tr>
<tr>
<td>DeleteFile (see page 173)</td>
<td>Deletes a file on the instrument file system.</td>
</tr>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>Export (see page 179)</td>
<td>Exports data to a file on the instrument file system.</td>
</tr>
<tr>
<td>ExportEx (see page 180)</td>
<td>Exports data to a file on the instrument file system.</td>
</tr>
<tr>
<td>GetProbeByName (see page 200)</td>
<td>Given a probe name, returns its corresponding probe object.</td>
</tr>
<tr>
<td>GetModuleByName (see page 196)</td>
<td>Given a module name, returns its corresponding hardware module object.</td>
</tr>
<tr>
<td>GetToolByName (see page 205)</td>
<td>Given a tool name, returns its corresponding tool object.</td>
</tr>
<tr>
<td>GetWindowByName (see page 206)</td>
<td>Given a window name, returns its corresponding window object.</td>
</tr>
<tr>
<td>Import (see page 212)</td>
<td>Imports data from a file located on the instrument file system.</td>
</tr>
<tr>
<td>ImportEx (see page 213)</td>
<td>Imports data from a file located on the instrument file system into a particular module.</td>
</tr>
<tr>
<td>GoOffline (see page 207)</td>
<td>Disconnects the user interface from the logic analyzer frame.</td>
</tr>
<tr>
<td>GoOnline (see page 211)</td>
<td>Connects the user interface to a specific logic analyzer frame.</td>
</tr>
<tr>
<td>IsOnline (see page 214)</td>
<td>Tells whether the user interface is connected to a logic analyzer frame.</td>
</tr>
<tr>
<td>New (see page 215)</td>
<td>Creates a new instrument Overview.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open (see page 216)</td>
<td>Loads a previously saved configuration file on the instrument file system.</td>
</tr>
<tr>
<td>PanelLock (see page 216)</td>
<td>Disables user access to the instrument front panel or remote display.</td>
</tr>
<tr>
<td>PanelUnlock (see page 220)</td>
<td>Re-enables user access to the instrument front panel or remote display.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
<tr>
<td>Run (see page 227)</td>
<td>Starts running all modules.</td>
</tr>
<tr>
<td>Save (see page 228)</td>
<td>Saves the current configuration to a file on the instrument file system.</td>
</tr>
<tr>
<td>Stop (see page 236)</td>
<td>Stops all currently running modules.</td>
</tr>
<tr>
<td>VBAFrameHelpTopic (see page 237)</td>
<td>Displays the help page and topic for an installed VBA project.</td>
</tr>
<tr>
<td>VBARunMacro (see page 237)</td>
<td>Runs the specified VBA macro as if that macro was selected in the Macros dialog box.</td>
</tr>
<tr>
<td>VBARunRPICommand (see page 238)</td>
<td>Runs an ASCII RPI command in VBA.</td>
</tr>
<tr>
<td>WaitComplete (see page 238)</td>
<td>Waits until all module, tool, and viewer measurements are complete.</td>
</tr>
</tbody>
</table>

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames (see page 263)</td>
<td>Gets a collection of logic analyzer frames connected via the multiframe connector.</td>
</tr>
<tr>
<td>Markers (see page 267)</td>
<td>Gets a collection of all the markers in the instrument.</td>
</tr>
<tr>
<td>Model (see page 268)</td>
<td>Gets the model number.</td>
</tr>
<tr>
<td>Modules (see page 268)</td>
<td>Gets a collection of all the hardware modules in the instrument.</td>
</tr>
<tr>
<td>Overview (see page 271)</td>
<td>Gets the XML-format Overview window specification.</td>
</tr>
<tr>
<td>PanelLocked (see page 272)</td>
<td>Indicates the front panel is locked. If locked, the message is returned.</td>
</tr>
<tr>
<td>Probes (see page 274)</td>
<td>Gets a collection of all currently defined probes.</td>
</tr>
<tr>
<td>RemoteComputerName (see page 274)</td>
<td>Gets or sets the remote computer name.</td>
</tr>
<tr>
<td>RemoteUserName (see page 275)</td>
<td>Gets or sets the remote user login name.</td>
</tr>
<tr>
<td>SelfTest (see page 277)</td>
<td>Gets the SelfTest object.</td>
</tr>
</tbody>
</table>
Markers Object

To Access
- Markers.Item (see page 266) IndexOrName
- Markers IndexOrName

Methods
There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments (see page 255)</td>
<td>Gets or sets the marker comments.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the marker.</td>
</tr>
<tr>
<td>TextColor (see page 283)</td>
<td>Gets or sets the marker text color.</td>
</tr>
<tr>
<td>BackgroundColor (see page 247)</td>
<td>Gets or sets the marker background color.</td>
</tr>
<tr>
<td>Position (see page 272)</td>
<td>Gets or sets the marker position.</td>
</tr>
</tbody>
</table>

See Also
- Markers (see page 123) object

Markers Object

To Access
- Instrument.Markers (see page 267)
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (see page 169)</td>
<td>Adds a new marker to the collection using specific values.</td>
</tr>
<tr>
<td>AddXML (see page 170)</td>
<td>Adds multiple markers to the collection using an XML string.</td>
</tr>
<tr>
<td>Remove (see page 225)</td>
<td>Removes a marker from the collection.</td>
</tr>
<tr>
<td>RemoveAll (see page 225)</td>
<td>Removes all markers from the collection.</td>
</tr>
<tr>
<td>RemoveXML (see page 225)</td>
<td>Removes multiple markers from the collection using an XML string.</td>
</tr>
</tbody>
</table>

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of markers in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the markers in the collection given either an index or name.</td>
</tr>
</tbody>
</table>

## Remarks

Window specific markers such as "Beginning of Data", "End of Data", and "Trigger" and markers with position other than by time are not part of the collection.

## See Also
- Markers (see page 267) property

## Markers Example

Visual Basic

```
' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Add a marker and set its position.
myInst.Markers.Add "Loc4", , , 0.00000001

' Change a marker's position and color.
Dim myMarker As AgtLA.Marker
Set myMarker = myInst.Markers("Loc4")
myMarker.Position = 0.000000005
myMarker.BackgroundColor = &HFF00
myMarker.TextColor = &HFF
```
' Add multiple markers to the collection using an XML string.
myInst.Markers.AddXML "<Markers>" + _
 "<Marker Name='XML M1' Comments='My Marker' " + _
 "ForegroundColor='hff00ff' BackgroundColor='h00ffff' " + _
 "Position='10 ns' LockPosition='T' />"> + _
 "<Marker Name='XML M2' ForegroundColor='hff00ff' " + _
 "BackgroundColor='h00ffff' Position='15 ns' " + _
 "LockPosition='F' />"> + _
 "<Marker Name='XML M3' BackgroundColor='h00ffff' " + _
 "Position='20 ns' LockPosition='T' />"> + _
 "<Marker Name='XML M4' Position='25 ns' LockPosition='F' />"> + _
 "<Marker Name='XML M5' LockPosition='T' />"> + _
 "<Marker Name='XML M6' />"> + _
 "</Markers>"

' Display all of the markers.
Dim myMarkerNames As String
For Each myMarker In myInst.Markers
 ' Add the marker name to the string.
 myMarkerNames = myMarkerNames + vbCrLf + myMarker.Name
Next
MsgBox "Marker names: " + myMarkerNames

' Remove a marker from the collection.
myInst.Markers.Remove "Loc4"

' Remove multiple markers from the collection using an XML string.
myInst.Markers.RemoveXML "<Markers>" + _
 "<Marker Name='XML M1' />"> + _
 "<Marker Name='XML M3' />"> + _
 "<Marker Name='XML M5' />"> + _
 "</Markers>"

' Remove all markers from the collection.
myInst.Markers.RemoveAll

Visual C++

// This simple Visual C++ Console application demonstrates how to use
the Agilent 168x/169x/169xx COM interface to set up markers.

// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path
in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
```c
#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[]) {
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr)) {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
            printf("Connecting to instrument '%s'\n", hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect = AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostname);

            // Run the measurement, wait for it to complete.
            pInst->Run(FALSE);
            pInst->WaitComplete(20);

            // Get the markers object.
            AgtLA::IMarkersPtr pMarkers = pInst->GetMarkers();

            // Add a marker and set its position.
            pMarkers->Add("Loc4", 0x000000, 0xffff00, 0.00000001);

            // Change a marker's position and color.
            AgtLA::IMarkerPtr pMarker = pMarkers->GetItem("Loc4");
            pMarker->PutPosition(0.000000005);
            pMarker->PutBackgroundColor(0xFF00);
            pMarker->PutTextColor(0xFF);

            // Add multiple markers to the collection using an XML string.
            _bstr_t myAddMarkers = "<Markers> \\
                             <Marker Name='XML M1' Comments='My Marker' \\
                              ForegroundColor='hff00ff' BackgroundColor='h00ffff' \\
                              Position='10 ns' LockPosition='T' /> \\
                             </Markers>";
```
<Marker Name='XML M2' ForegroundColor='hff00ff' \ 
  BackgroundColor='h00ffff' Position='15 ns' \ 
  LockPosition='F' /> \
<Marker Name='XML M3' BackgroundColor='h00ffff' \ 
  Position='20 ns' LockPosition='T' /> \
<Marker Name='XML M4' Position='25 ns' \ 
  LockPosition='F' /> \
<Marker Name='XML M5' LockPosition='T' /> \
<Marker Name='XML M6' /> \
</Markers>);
pMarkers->AddXML(myAddMarkers);

// Display all of the markers.
for (long i = 0; i < pMarkers->GetCount(); i++)
{
  pMarker = pMarkers->GetItem(i);
  _bstr_t name = pMarker->GetName();
  printf("Marker name = %s\n", (char*) name);
}

// Remove a marker from the collection.
pMarkers->Remove("Loc4");
// Remove multiple markers from the collection using an XML
// string.
_bstr_t myRemoveMarkers = "<Markers> \
<Marker Name='XML M1' /> \
<Marker Name='XML M3' /> \
<Marker Name='XML M5' /> \
</Markers>";
pMarkers->RemoveXML(myRemoveMarkers);

// Remove all markers from the collection.
pMarkers->RemoveAll();
}
catch (_com_error& e) {
  DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
else
{
  printf("CoInitialize failed\n");
}
return 0;

// Displays the last error -- used to show the last exception
// information.
//
// void DisplayError(_com_error& error) {

```
printf("\n")

printf("Fatal Unexpected Error:\n")
printf(" Error Number = %08lx\n", error.Error());

static char errorStr[1024];
_bstr_t desc = error.Description();

if (desc.length() == 0)
{
    // Don’t have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
    {
        errorStr[nLen - 2] = '\0';
    }
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);

Module Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

To Access

- Instrument.GetModuleByName(Name) (see page 196)
- Modules.Item (see page 266) IndexOrName
- Modules IndexOrName

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
<tr>
<td>WaitComplete (see page 238)</td>
<td>Waits until the module’s measurement completes.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusSignals (see page 250)</td>
<td>Gets a collection of the module’s defined bus/signals.</td>
</tr>
<tr>
<td>Description (see page 260)</td>
<td>Gets a description of the module.</td>
</tr>
<tr>
<td>Frame (see page 263)</td>
<td>Gets the frame in which the module resides.</td>
</tr>
<tr>
<td>Model (see page 268)</td>
<td>Gets the model number.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the module.</td>
</tr>
<tr>
<td>RunningStatus (see page 275)</td>
<td>Gets the detailed running status of the module.</td>
</tr>
<tr>
<td>Slot (see page 278)</td>
<td>Gets the module’s slot location in the frame.</td>
</tr>
<tr>
<td>Status (see page 280)</td>
<td>Gets the status of the module.</td>
</tr>
<tr>
<td>StatusMsg (see page 281)</td>
<td>Gets the formatted status message.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the specific module type.</td>
</tr>
</tbody>
</table>

### See Also
- Modules (see page 129) object

### Modules Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

#### To Access
- Instrument.Modules (see page 268)

#### Methods
There are no methods.

#### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of modules in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the modules in a collection given either a slot, index, or name.</td>
</tr>
</tbody>
</table>

### See Also
- Modules (see page 268) property
- Module (see page 128) object
- GetModuleByName (see page 196) method

### PattgenModule Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 131) ]
To Access

```
Dim variable As AgtLA.PattgenModule
Set variable = Module (see page 128)
```

Description

The **PattgenModule** object represents a pattern generator hardware module.

Since this object is derived from the **Module** (see page 128) object, a **Set** must be used to get to these specific methods and properties. For example:

```
' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Get the PattgenModule specific object.
Dim myPattgen As AgtLA.PattgenModule
Set myPattgen = myInst.Modules(0)
MsgBox "Number of lines in main sequence: " + myPattgen.NumLines
```

<table>
<thead>
<tr>
<th><strong>Method</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GetLine (see page 194)</td>
<td>Gets an instruction or vector at line number.</td>
</tr>
<tr>
<td>GetLineLabel (see page 196)</td>
<td>Gets a vector's label value at line number.</td>
</tr>
<tr>
<td>InsertLine (see page 214)</td>
<td>Inserts a new instruction or vector after line number.</td>
</tr>
<tr>
<td>RemoveLine (see page 226)</td>
<td>Removes the instruction or vector at line number.</td>
</tr>
<tr>
<td>Reset (see page 226)</td>
<td>Resets the current line number to the first line.</td>
</tr>
<tr>
<td>Resume (see page 227)</td>
<td>Resumes running the pattern generator from the current line number.</td>
</tr>
<tr>
<td>Run (see page 228)</td>
<td>Starts running the pattern generator.</td>
</tr>
<tr>
<td>SetLine (see page 230)</td>
<td>Sets an instruction or vector at line number.</td>
</tr>
<tr>
<td>SetLineLabel (see page 230)</td>
<td>Sets a vector's label value at line number.</td>
</tr>
<tr>
<td>Step (see page 235)</td>
<td>Steps the pattern generator from the current line number.</td>
</tr>
<tr>
<td>Stop (see page 236)</td>
<td>Stops the pattern generator if it is currently running.</td>
</tr>
</tbody>
</table>
(Also Includes Module (see page 128) object methods)

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumLines (see page 270)</td>
<td>Gets the number of lines in the main sequence.</td>
</tr>
</tbody>
</table>

(Also Includes Module (see page 128) object properties)

**PattgenModule Example**

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Load the configuration file.
myInst.Open ("C:\LA\Configs\pattgen.ala")

' Get the PattgenModule specific object.
Dim moduleName As String
moduleName = "My 16720A-1"
Dim myPattgen As AgtLA.PattgenModule
Set myPattgen = myInst.GetModuleByName(moduleName)
MsgBox moduleName + " number of lines in main sequence: " + Str(myPattgen.NumLines)

' Display the pattern generator instruction or vector at a particular line number.
Dim myInstructionOrVector As String
myInstructionOrVector = myPattgen.GetLine(24)
MsgBox moduleName + " instruction/vector at line 24: " + myInstructionOrVector

' Set the instruction or vector at a line number.
myInstructionOrVector = "Break"
myPattgen.SetLine 30, myInstructionOrVector

' Display the pattern generator label value at a line number.
Dim myLineLabel As String
Dim myLabelValue As String
myLabelValue = myPattgen.GetLineLabel(24, "My Bus 1")
MsgBox moduleName + " My Bus 1 value at line 24: " + myLabelValue

' Set the label value at a line number.
myLabelValue = "haa"
myPattgen.SetLineLabel 27, "My Bus 1", myLabelValue
MsgBox moduleName + " My Bus 1 value set at line 27: " + myLabelValue
' Insert a new instruction or vector after a line number.
myPattgen.InsertLine 1, "Vector 'My Bus 1' = h11"
MsgBox moduleName + " line inserted after vector 1"

' Remove a range of lines.
myPattgen.RemoveLine ("5..12")
MsgBox moduleName + " vectors from lines 5 through 12 removed"

' Remove all lines.
myPattgen.RemoveLine ("All")
MsgBox moduleName + " all vectors removed"

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to control a pattern
// generator module.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// instdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \\n// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

////////////////////////////////////////////////////////////////////
// Forward declarations.
////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& err);

////////////////////////////////////////////////////////////////////
// main() entry point.
////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
  printf("*** Main()\n");

  //
  // Initialize the Microsoft COM/ActiveX library.
  //
  HRESULT hr = CoInitialize(0);
if (SUCCEEDED(hr))
{
    try { // Catch any unexpected run-time errors.
        _bstr_t hostname = "mtx33"; // TODO, use your logic
        // analysis system hostname.
        printf("Connecting to instrument '%s'\n", (char*) hostname);

        // Create the connect object and get the instrument object.
        AgtLA::IConnectPtr pConnect =
            AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
        AgtLA::IInstrumentPtr pInst =
            pConnect->GetInstrument(hostname);

        // Load the configuration file.
        _bstr_t configFile = "C:\LA\Configs\pattgen.ala";
        printf("Loading the config file '%s'\n", (char*) configFile);
        pInst->Open(configFile, FALSE, "", TRUE);

        // Get the PattgenModule specific object.
        _bstr_t moduleName = "My 16720A-1";
        AgtLA::IPattgenModulePtr pPattgen =
            pInst->GetModuleByName(moduleName);

        // Display the pattern generator instruction or vector
        // at a particular line number.
        _bstr_t myInstructionOrVector;
        myInstructionOrVector = pPattgen->GetLine(24);
        printf("%s instruction/vector at line 24 = '%s'\n",
               (char *) moduleName, (char*) myInstructionOrVector);

        // Set the instruction or vector at a line number.
        myInstructionOrVector = "Break";
        pPattgen->SetLine(30, myInstructionOrVector);

        // Display the pattern generator label value at a line number.
        _bstr_t myLabelValue;
        myLabelValue = pPattgen->GetLineLabel(24, "My Bus 1");
        printf("%s My Bus 1 value at line 24 = '%s'\n",
               (char *) moduleName, (char*) myLabelValue);

        // Set the label value at a line number.
        myLabelValue = "haa";
        pPattgen->SetLineLabel(27, "My Bus 1", myLabelValue);
        printf("%s My Bus 1 value set at line 27 = '%s'\n",
               (char *) moduleName, (char*) myLabelValue);

        // Insert a new instruction or vector after a line number.
        pPattgen->InsertLine(1, "Vector 'My Bus 1' = h11");
        printf("%s line inserted after vector 1\n", (char *) moduleName);

        // Remove a range of lines.
        pPattgen->RemoveLine("5..12");
        printf("%s vectors from lines 5 through 12 removed\n", (char *) moduleName);
// Remove all lines.
pPattgen->RemoveLine("All");
printf("%s all vectors removed\n", (char *) moduleName);
}
catch (_com_error& e) {
  DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
else {
  printf("CoInitialize failed\n");
}
return 0;
}

////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.

void DisplayError(_com_error& error) {
  printf("*** DisplayError()\n");

  printf("Fatal Unexpected Error:\n");
  printf(" Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();

  if (desc.length() == 0) {
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd)) {
      errorStr[nLen - 2] = '\0';
    }
  } else {
    strcpy(errorStr, desc);
  }

  printf(" Error Message = %s\n", (char*) errorStr);
}
Probe Object

To Access
- Instrument.probeByName(see page 200)
- Probes.Item (see page 266) IndexOrName
- Probes IndexOrName

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the probe.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the probe’s type.</td>
</tr>
</tbody>
</table>

See Also
- Probes (see page 135) object

Probes Object

To Access
- Instrument.probes (see page 274)

Methods
There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of probes in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the probes in the collection given either an index or name.</td>
</tr>
</tbody>
</table>

See Also
- Probes (see page 274) property
- Probe (see page 135) object
• GetProbeByName (see page 200) method

Probes Example

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Display all of the probe names.
Dim myProbeNames As String
Dim myProbe As AgtLA.Probe

For Each myProbe in myInst.Probes
    ' Add the probe's name to the string.
    myProbeNames = myProbeNames + vbCrLf + myProbe.Name
Next
MsgBox "Probe names: " + myProbeNames

**Visual C++**

// This simple Visual C++ Console application demonstrates how to use the Agilent 168x/169x/169xx COM interface to display all the probe names.

// This project was created in Visual C++ Developer. To create a similar project:

// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \ COM Automation/agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect to (search for "TODO" below).

#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& err);
int main(int argc, char* argv[]) {
    printf("*** Main()\n\n");
    
    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr)) {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
            printf("Connecting to instrument '%s'\n", (char*) hostname);
            
            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);
            
            // Load the configuration file.
            _bstr_t configFile = "C:\\LA\\Configs\\probes.ala";
            printf("Loading the config file '%s'\n", (char*) configFile);
            pInst->Open(configFile, FALSE, "", TRUE);
            
            // Display all of the probe names.
            AgtLA::IProbesPtr pProbes = pInst->GetProbes();
            for (long i = 0; i < pProbes->GetCount(); i++) {
                AgtLA::IProbePtr pProbe = pProbes->GetItem(i);
                _bstr_t name = pProbe->GetName();
                printf("Probe name = '%s'\n", (char*) name);
            }
        }
    }
    catch (_com_error& e) {
        DisplayError(e);
    }
    
    // Uninitialize the Microsoft COM/ActiveX library.
    CoUninitialize();
}
else {
    printf("CoInitialize failed\n");
}
return 0;
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);
        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }

    printf(" Error Message = %s\n", (char*) errorStr);
}

SampleBusSignalData Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 139) ]

To Access

* Dim variable As AgtLA.SampleBusSignalData
  Set variable = BusSignalData (see page 108)

Description

The SampleBusSignalData object represents the data associated with a bus/signal. The data can be uploaded using the methods GetDataBySample (see page 189) and GetDataByTime (see page 192).

Since this object is derived from the BusSignalData (see page 108) object, a Set must be used to get to these specific methods and properties. For example:

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see
page 118) and use it
' to access the Instrument object. In this example, "myInst"
' represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)"
' (in the online help),
' the Instrument object is already created and is globally
' accessible using "AgtLA". In this example, substitute "myInst"
' with "AgtLA" to access the global Instrument object in VBA.
'
' Get the AnalyzerModule specific object.
Dim myData As AgtLA.SampleBusSignalData
Set myData = myInst.Modules(0).BusSignals(0).BusSignalData
MsgBox "Sample Range: " + myData.StartSample + "." + myData.EndSample

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDataBySample</td>
<td>Given a sample range, returns an array of data.</td>
</tr>
<tr>
<td>GetDataByTime</td>
<td>Given a time range, returns an array of data.</td>
</tr>
<tr>
<td>GetNumSamples</td>
<td>Given a range, returns the number of samples stored.</td>
</tr>
<tr>
<td>GetSampleNumByTime</td>
<td>Gets the closest sample number corresponding to the time given.</td>
</tr>
<tr>
<td>GetTime</td>
<td>Given a range, returns the time for this bus/signal in the format specified by the data type given.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataT ype</td>
<td>Gets the recommended bus/signal data type.</td>
</tr>
<tr>
<td>EndSample</td>
<td>Gets the data's ending sample number relative to trigger.</td>
</tr>
<tr>
<td>EndTime</td>
<td>Gets the data’s ending time relative to trigger.</td>
</tr>
<tr>
<td>StartSample</td>
<td>Gets the data’s starting sample number relative to trigger.</td>
</tr>
<tr>
<td>StartTime</td>
<td>Gets the data’s starting time relative to trigger.</td>
</tr>
</tbody>
</table>

SampleBusSignalData Example

Visual Basic
' When using Visual Basic outside of the Agilent Logic Analyzer
' application, you must create the Connect object (see page 118) and use it
' to access the Instrument object. In this example, "myInst"
' represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)"
' (in the
online help),

' the Instrument object is already created and is globally
' accessible using "AgtLA". In this example, substitute "myInst"
' with "AgtLA" to access the global Instrument object in VBA.

' Run the measurement, wait for it to complete.
myInst.Run
myInst.WaitComplete (20)

' Display all of the bus/signal data.
Dim myString As String
Dim printHeader As Boolean
Dim myBusSignals As AgtLA.BusSignals
Set myBusSignals = myInst.GetModuleByName("My 1690A-1").BusSignals
Dim myBusSignal As AgtLA.BusSignal
Dim myData As AgtLA.SampleBusSignalData

Dim myNumDataRows As Long
Dim myStartSample As Long
Dim myEndSample As Long
Dim i As Long

printHeader = True
myStartSample = -5 ' Sample range to upload.
myEndSample = 5

For Each myBusSignal In myBusSignals
   Set myData = myBusSignal.BusSignalData
   If printHeader = True Then
      myString = myString + "Sample range: " + 
               Str(myData.StartSample) + "." + Str(myData.EndSample)
      myString = myString + ", Time range: " + 
               Str(myData.StartTime) + "." + Str(myData.EndTime) + 
               vbCrLf
      printHeader = False
   End If

   ' Print the bus/signal information.
   myString = myString + vbCrLf + "Name: " + myBusSignal.Name
   myString = myString + ", BitSize=" + Str(myBusSignal.BitSize) + 
               ", ByteSize=" + Str(myBusSignal.ByteSize) + vbCrLf

   ' Print the bus/signal data.
   Select Case myBusSignal.BusSignalType
   Case AgtBusSignalSampleNum
      Dim lArray() As Long
      lArray = myData.GetDataBySample(myStartSample, myEndSample, _
                                      AgtDataLong, myNumDataRows)
      For i = 0 To myNumDataRows - 1
         myString = myString + Str(lArray(i)) + " 
      Next i
   Case AgtBusSignalTime
      Dim dArray() As Double
      dArray = myData.GetDataBySample(myStartSample, myEndSample, _
                                       AgtDataTime, myNumDataRows)
      For i = 0 To myNumDataRows - 1
myString = myString + Str(dArray(i)) + " 
Next i
Case AgtBusSignalGenerated
' Decimal holds a maximum of 96 bits unsigned.
Dim vArray As Variant
  vArray = myData.GetDataBySample(myStartSample, myEndSample, _
            AgtDataDecimal, myNumDataRows)
For i = 0 To myNumDataRows - 1
  myString = myString + Str(vArray(i)) + " 
Next i
Case AgtBusSignalProbed
  ' Use the data type that is appropriate for your bus width:
  ' - AgtDataLong holds a maximum of 31 bits unsigned.
  ' - AgtDataDouble holds a maximum of 52 bits unsigned.
  ' - AgtDataDecimal holds a maximum of 96 bits unsigned.
  lArray = myData.GetDataBySample(myStartSample, myEndSample, _
            AgtDataLong, myNumDataRows)
For i = 0 To myNumDataRows - 1
  myString = myString + Hex$(lArray(i)) + " 
Next i
End Select
Next
MsgBox myString

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to display captured data.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////

void DisplayRawData(
  _bstr_t& busSignalName,
  _variant_t& rawDataArray,
  long numBytesPerRow);
```c
void DisplayBusSignalData(
    _bstr_t& busSignalName,
    _variant_t& busSignalArray);
void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
    printf("*** Main()\n");
    
    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
            printf("Connecting to instrument 's\n", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\\LA\\Configs\\config.ala";
            printf("Loading the config file 's\n", (char*) configFile);
            pInst->Open(configFile, FALSE, "", TRUE);

            // Run the measurement, wait for it to complete.
            pInst->Run(FALSE);
            pInst->WaitComplete(20);

            // Get data from the Listing window.
            _bstr_t windowName = "Listing-1";
            AgtLA::IWindowPtr pWindow = pInst->GetWindowByName(windowName);

            // For each bus/signal, display a range of data.
            long start = -10;
            long end = 10;
            _variant_t data;
            long numRowsRet;
            long numBytesPerRow;
            AgtLA::IBusSignalsPtr pBusSignals = pWindow->GetBusSignals();
            printf("\n");
            for (long i = 0; i < pBusSignals->GetCount(); i++)
            {
                // Get the data for the bus/signal.
AgtLA::IBusSignalPtr pBusSignal = pWindow->GetBusSignals()->GetItem(i);
_bstr_t busSignal = pBusSignal->GetName();
printf("Bus/signal: '%s'\n", (char*) busSignal);
AgtLA::ISampleBusSignalDataPtr pSampleData = pBusSignal->GetBusSignalData();

switch(pBusSignal->GetBusSignalType())
{
  case AgtLA::AgtBusSignalProbed:
  {
    printf(" Type: 'AgtBusSignalProbed'\n");
    // "raw" and "long" formats supported.
    if (pBusSignal->GetBitSize() > 32) {
      data = pSampleData->GetDataBySample(start, end,
        AgtLA::AgtDataRaw, &numRowsRet);
      numBytesPerRow = pBusSignal->GetByteSize();
      printf(" Data type: 'AgtDataRaw'\n");
      printf("(%d bytes/row)\n", numBytesPerRow);
      DisplayRawData(busSignal, data, numBytesPerRow);
    }
    else {
      data = pSampleData->GetDataBySample(start, end,
        AgtLA::AgtDataLong, &numRowsRet);
      printf(" Data type: 'AgtDataLong'\n");
      DisplayBusSignalData(busSignal, data);
    }
  }
  break;

  case AgtLA::AgtBusSignalGenerated:
  {
    printf(" Type: 'AgtBusSignalGenerated'\n");
    data = pSampleData->GetDataBySample(start, end,
      AgtLA::AgtDataStringHex, &numRowsRet);
    printf(" Data type: 'AgtDataStringHex'\n");
    DisplayBusSignalData(busSignal, data);
  }
  break;

  case AgtLA::AgtBusSignalSampleNum:
  {
    printf(" Type: 'AgtBusSignalSampleNum'\n");
    data = pSampleData->GetDataBySample(start, end,
      AgtLA::AgtDataLong, &numRowsRet);
    printf(" Data type: 'AgtDataLong'\n");
    DisplayBusSignalData(busSignal, data);
  }
  break;

  case AgtLA::AgtBusSignalTime:
  {
    printf(" Type: 'AgtBusSignalTime'\n");
    data = pSampleData->GetDataBySample(start, end,
      AgtLA::AgtDataTime, &numRowsRet);
    printf(" Data type: 'AgtDataTime'\n");
    DisplayBusSignalData(busSignal, data);
  }
  break;
}
```c
// Displays the data in raw data format.
//
void DisplayRawData(_bstr_t& busSignalName,
    _variant_t& varArray,
    long numBytesPerRow)
{
    long numSamples;
    long lBound;
    HRESULT hr = SafeArrayGetLBound(varArray.parray, 1, &lBound);
    if (SUCCEEDED(hr))
    {
        long uBound;
        hr = SafeArrayGetUBound(varArray.parray, 1, &uBound);
        if (SUCCEEDED(hr))
        {
            printf(" Variant data format: VT_UI1 (unsigned char)\n");
            byte* pByteArray;
            hr = SafeArrayAccessData(varArray.parray,
                (void**) &pByteArray);
            if (SUCCEEDED(hr))
            {
                numSamples = (uBound - lBound + 1) / numBytesPerRow;
                byte* pByte = pByteArray;
                for (int i = 0; i < numSamples; i++)
                {
                    printf(" dataArray[%d]: ", i);
                    printf(" \n");
                }
            }
        }
    }
}
```
for (int j = 0; j < numBytesPerRow; j++)
{
    printf("%02x ", pByte[j]);
}

pByte += numBytesPerRow;
printf("\n");
}

printf("\n");
SafeArrayUnaccessData(varArray.parray);

/////////////////////////////////////////////////////////////////////
// Displays bus/signal data in the given array.
//
void DisplayBusSignalData(_bstr_t& busSignalName,
    _variant_t& varArray)
{
    signed _int8* pArrayInt8 = NULL;
    unsigned _int8* pArrayUInt8 = NULL;
    signed _int16* pArrayInt16 = NULL;
    unsigned _int16* pArrayUInt16 = NULL;
    signed _int32* pArrayInt32 = NULL;
    unsigned _int32* pArrayUInt32 = NULL;
    signed _int64* pArrayInt64 = NULL;
    unsigned _int64* pArrayUInt64 = NULL;
    double* pArrayDouble = NULL;
    BSTR* pArrayBstr = NULL;

    SAFEARRAY* pDataArray = varArray.parray;

    long lBound;
    HRESULT hr = SafeArrayGetLBound(pDataArray, 1, &lBound);
    if (FAILED(hr))
    {
        printf("SafeArrayGetLBound failed\n");
    }

    long uBound;
    hr = SafeArrayGetUBound(pDataArray, 1, &uBound);
    if (FAILED(hr))
    {
        printf("SafeArrayGetUBound failed\n");
    }

    long numSamples = uBound - lBound + 1;

    switch (varArray.vt - VT_ARRAY)
case VT_I1:
{
    printf(" Variant data format: VT_I1 (char)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayInt8);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %02cx\n", i, pArrayInt8[i]);
    }
    break;
}

case VT_UI1:
{
    printf(" Variant data format: VT_UI1 (unsigned char)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayUInt8);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %02cx\n", i, pArrayUInt8[i]);
    }
    break;
}

case VT_I2:
{
    printf(" Variant data format: VT_I2 (short)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayInt16);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %04hx\n", i, pArrayInt16[i]);
    }
    break;
}

case VT_UI2:
{
    printf(" Variant data format: VT_UI2 (unsigned short)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayUInt16);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %04hx\n", i, pArrayUInt16[i]);
    }
    break;
}

case VT_I4:
{
    printf(" Variant data format: VT_I4 (long)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayInt32);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %08lx\n", i, pArrayInt32[i]);
    }
}
break;
}

case VT_UI4:
{
    printf(" Variant data format: VT_UI4 (unsigned long)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayUInt32);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %08lx\n", i, pArrayUInt32[i]);
    }
    break;
}

case VT_I8:
{
    printf(" Variant data format: VT_I8 (__int64)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayInt64);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %016I64x\n", i, pArrayInt64[i]);
    }
    break;
}

case VT_UI8:
{
    printf(" Variant data format: VT_UI8 (unsigned __int64)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayUInt64);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %016I64x\n", i, pArrayUInt64[i]);
    }
    break;
}

case VT_R8:
{
    printf(" Variant data format: VT_R8 (double)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayDouble);
    for (int i = lBound; i <= uBound; i++)
    {
        printf(" dataArray[%d]: %0le\n", i, pArrayDouble[i]);
    }
    break;
}

case VT_BSTR:
{
    printf(" Variant data format: VT_BSTR (_bstr_t)\n");
    hr = SafeArrayAccessData(pDataArray, (void**) &pArrayBstr);
    for (int i = lBound; i <= uBound; i++)
{  printf("  dataArray[%d]: %S\n", i, pArrayBstr[i]);
  break;
}

default:
{  printf("  Variant data format: unknown\n");
   hr = E_FAIL;
   break;
}

if (FAILED(hr))
{  printf("SafeArrayAccessData failed\n");
}

printf("\n");
SafeArrayUnaccessData(pDataArray);

/////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.
/////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& error)
{
  printf("*** DisplayError()\n");

  printf("Fatal Unexpected Error:\n");
  printf("  Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();

  if (desc.length() == 0)
  {  // Don't have a description string.
     strcpy(errorStr, error.ErrorMessage());
     int nLen = lstrlen(errorStr);

     // Remove funny carriage return ctrl<M>.
     if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
     {  errorStr[nLen - 2] = '\0';
     }
   }
  else
  {  strcpy(errorStr, desc);
  }
printf(" Error Message = %s\n", (char*) errorStr);
}

SampleDifference Object

To Access

- SampleDifferences.Item (see page 266) IndexOrName
- SampleDifferences IndexOrName

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusSignalDifferences</td>
<td>Gets a collection of all the buses/signals with differences for this sample.</td>
</tr>
<tr>
<td>SampleNum</td>
<td>Gets the sample number at which differences occurred.</td>
</tr>
</tbody>
</table>

See Also

- SampleDifferences (see page 149) object

SampleDifferences Object

To Access

- CompareWindow.SampleDifferences (see page 276)

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count</td>
<td>Gets the number of bus/signal differences in the collection.</td>
</tr>
<tr>
<td>Item</td>
<td>Given an index into the collection, gets a SampleDifference (see page 149)</td>
</tr>
</tbody>
</table>

See Also

- SampleDifferences (see page 276) property

SelfTest Object

SelfTest Object
To Access

- Instrument(SelfTest (see page 277)

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestAll (see page 237)</td>
<td>Runs an instrument’s self-tests.</td>
</tr>
</tbody>
</table>

Properties

There are no properties.

See Also

- SelfTest (see page 277) property

SelfTest Example

The following script runs all of the tests available on the target instrument:

Visual Basic

```vbnet
Dim result As String
result = theInstrument.SelfTest.TestAll()
```

Visual C++

```cpp
// This simple Visual C++ Console application demonstrates how to
// use the Agilent 168x/169x/169xx COM interface to run the logic
// analysis system's self tests.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA\COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"
```

```cpp
//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);
```

```cpp
//////////////////////////////////////////////////////////////////////
// main() entry point.
//////////////////////////////////////////////////////////////////////
```
int main(int argc, char* argv[]) 
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "col-mil20"; // TODO, use your logic
            printf("Connecting to instrument '%s'
", (char*) hostname);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Run the logic analysis system self tests and print the
            // results.
            AgtLA::ISelfTestPtr pSelfTest = pInst->GetSelfTest();
            _bstr_t testResults = pSelfTest->TestAll();
            printf("Self test results: %s
", (char*) testResults);
        }
        catch (_com_error& e) {
            DisplayError(e);
        }

        // Uninitialize the Microsoft COM/ActiveX library.
        CoUninitialize();
    } else
    {
        printf("CoInitialize failed\n");
    }

    return 0;
}

/////////////////////////////////////////////////////////////////////////

// Displays the last error -- used to show the last exception
// information.
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");

    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

static char errorStr[1024];
_bstr_t desc = error.Description();

if (desc.length() == 0)
{
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl\M.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
    {
        errorStr[nLen - 2] = '\0';
    }
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);

---

**Tool Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 153) ]

**To Access**
- Instrument.GetToolByName(Name) (see page 205)
- Tools.Item (see page 266) IndexOrName
- Tools IndexOrName

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusSignals (see page 250)</td>
<td>Gets a collection of the tool's defined bus/signals.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the tool.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the specific tool type.</td>
</tr>
</tbody>
</table>

**See Also**
- Tools (see page 153) object
Tools Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 153) ]

To Access

- Instrument.Tools (see page 284)

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of tools in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the tools in the collection given either an index or name.</td>
</tr>
</tbody>
</table>

See Also

- Tools (see page 284) property
- Tool (see page 152) object
- GetToolByName (see page 205) method

Tools Example

Visual Basic

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help),
' the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Display all of the tool names.
Dim myToolNames As String
Dim myTool As AgtLA.Tool

For Each myTool in myInst.Tools
    ' Add the tool's name to the string.
    myToolNames = myToolNames + vbCrLf + myTool.Name
Next
MsgBox "Tool names: " + myToolNames

' Get the MPC8XX Inverse Assembler tool; then, execute the QueryCommand.
Dim XMLCmdResponse As String
If myInst.GetToolByName("Motorola PowerQUICC (MPC8XX) Inverse " + _
   "Assembler-1").QueryCommand("GetProperties", XMLCmdResponse) _
Then
MsgBox "MPC8XX IA Properties: " + XMLCmdResponse
End If

Visual C++
// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to display all the tool
// names and, with a specific tool, execute a QueryCommand to get the
// tool's properties.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// instdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& err);

//////////////////////////////////////////////////////////////////////
// main() entry point.
//////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
            printf("Connecting to instrument '%s'\n", (char*) hostname);

            // Create the connect object and get the instrument object.
AgtLA::IConnectPtr pConnect = 
    AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
AgtLA::IInstrumentPtr pInst = 
    pConnect->GetInstrument(hostname);

// Load the configuration file.
_bstr_t configFile = "C:/LA/Configs/config.ala";
printf("Loading the config file '%s'\n", (char*) configFile);
pInst->Open(configFile, FALSE, "", TRUE);

// Display all of the tool names.
AgtLA::IToolsPtr pTools = pInst->GetTools();
for (long i = 0; i < pTools->GetCount(); i++)
{
    AgtLA::IToolPtr pTool = pTools->GetItem(i);
    _bstr_t name = pTool->GetName();
    printf("Tool name = %s\n", (char*) name);
}

// Get the MPC8XX Inverse Assembler tool; then,
// execute the QueryCommand.
_bstr_t myTool = 
    "Motorola PowerQUICC (MPC8XX) Inverse Assembler-1";
AgtLA::IToolPtr pTool = pInst->GetToolByName(myTool);
BSTR cmdResponseXML;
if (pTool->QueryCommand("GetProperties", &cmdResponseXML)) {
    printf("MPC8XX IA Properties '%S'\n",
            (char*) cmdResponseXML);
}
SysFreeString(cmdResponseXML);
}

try {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
printf(" Error Number = %08lx\n", error.Error());

static char errorStr[1024];
_bstr_t desc = error.Description();

if (desc.length() == 0)
{
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
    {
        errorStr[nLen - 2] = '\0';
    }
}
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);

---

**VbaViewChart Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**To Access**
- VbaViewWindow.Chart (see page 253)

**Description**
The `VbaViewChart` object represents a chart in the VbaViewWindow.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw (see page 178)</td>
<td>Draws the chart.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis (see page 246)</td>
<td>Gets the chart axis given an axis type.</td>
</tr>
<tr>
<td>ChartType (see page 253)</td>
<td>Gets or sets the chart type.</td>
</tr>
<tr>
<td>Data (see page 260)</td>
<td>Gets the chart data.</td>
</tr>
<tr>
<td>HasLegend (see page 264)</td>
<td>Gets or sets if the legend is visible.</td>
</tr>
<tr>
<td>HasTitle (see page 264)</td>
<td>Gets or sets if the title is visible.</td>
</tr>
<tr>
<td>Legend (see page 267)</td>
<td>Gets the chart legend.</td>
</tr>
<tr>
<td>Title (see page 284)</td>
<td>Gets the title of the chart.</td>
</tr>
</tbody>
</table>
See Also • Chart (see page 253) property

VbaViewChartAxis Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

To Access • VbaViewChart.Axis (see page 246)

Description The VbaViewChartAxis object represents a chart in the VbaViewWindow.

Methods There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxisBase (see page 246)</td>
<td>Gets or sets the chart axis base.</td>
</tr>
<tr>
<td>BitSize (see page 248)</td>
<td>Gets or sets the width of the data in bits. This is used to format the Axis values.</td>
</tr>
<tr>
<td>HasTitle (see page 264)</td>
<td>Gets or sets if the title is visible.</td>
</tr>
<tr>
<td>Title (see page 284)</td>
<td>Gets the title of the axis.</td>
</tr>
</tbody>
</table>

See Also • Axis (see page 246) property

VbaViewChartData Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

To Access • VbaViewChart.Data (see page 260)

Description The VbaViewChartData object represents the data in a VbaViewChart.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddPointArrays (see page 170)</td>
<td>Adds an array of points to the chart. This is only valid for chart types AgtChartTypeLine and AgtChartTypeXYScatter.</td>
</tr>
<tr>
<td>Clear (see page 171)</td>
<td>Clears all of the chart data.</td>
</tr>
<tr>
<td>GetGroupCaption (see page 193)</td>
<td>Gets the caption associated with a group (row).</td>
</tr>
<tr>
<td>GetValueCaption (see page 206)</td>
<td>Gets the caption associated with all values at index (column).</td>
</tr>
<tr>
<td>SetGroupCaption (see page 229)</td>
<td>Sets the caption associated with a group (row).</td>
</tr>
<tr>
<td>SetValue (see page 231)</td>
<td>Sets an individual value in the chart array.</td>
</tr>
</tbody>
</table>
Properties

There are no properties.

See Also

- Data (see page 260) property

VbaViewChartFont Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

To Access

- VbaViewChartTitle.Font (see page 262)

Description

The VbaViewChartFont object represents the font of a VbaViewChartTitle.

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold (see page 248)</td>
<td>Gets or sets the text thickness.</td>
</tr>
<tr>
<td>Color (see page 254)</td>
<td>Gets or sets the text color.</td>
</tr>
<tr>
<td>FaceName (see page 262)</td>
<td>Gets or sets the text face name string.</td>
</tr>
<tr>
<td>Size (see page 278)</td>
<td>Gets or sets the text size.</td>
</tr>
</tbody>
</table>

See Also

- Font (see page 262) property

VbaViewChartLegend Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

To Access

- VbaViewChart.Legend (see page 267)

Description

The VbaViewChartLegend object represents the legend in a VbaViewChart.

Methods

There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position (see page 273)</td>
<td>Gets or sets the chart legend position.</td>
</tr>
</tbody>
</table>
See Also  • Legend (see page 267) property

**VbaViewChartTitle Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

To Access  • VbaViewChart.Title (see page 284)

Description The **VbaViewChartTitle** object represents the title of a VbaViewChart.

Methods There are no methods.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption (see page 251)</td>
<td>Gets or sets the chart title caption.</td>
</tr>
<tr>
<td>Font (see page 282)</td>
<td>Gets the chart title font.</td>
</tr>
</tbody>
</table>

See Also  • Title (see page 284) property

**VbaViewWebBrowser Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 159) ]

To Access  • VbaViewWindow.WebBrowser (see page 288)

Description The **VbaViewWebBrowser** object represents a web browser in the VbaViewWindow.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear (see page 171)</td>
<td>Displays an empty web page.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebBrowser (see page 288)</td>
<td>Gets the contained IWebBrowser2 interface.</td>
</tr>
</tbody>
</table>

Requirements  • **Version (see page 62):** 3.20 or later.

See Also  • WebBrowser (see page 288) property

**VbaViewWebBrowser Example**

Dim browser As SHDocVw.WebBrowser  ' "Microsoft Internet Controls" reference.
Set browser = myVbaViewWindow.WebBrowser.WebBrowser

' Display the Agilent home page.
browser.Navigate("http://www.agilent.com")

' You can optionally wait until the page is completely loaded.
While browser.readyState <> READYSTATE_COMPLETE
    DoEvents
Wend

**VbaViewWindow Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**To Access**

Dim variable As AgtLA.VbaViewWindow
Set variable = Window (see page 160)

**Description**
The VbaViewWindow object represents an output window for Visual Basic for Applications (VBA) programs.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearOutput (see page 172)</td>
<td>Clears the strings from the output window.</td>
</tr>
<tr>
<td>WriteOutput (see page 242)</td>
<td>Writes a string to the output window.</td>
</tr>
</tbody>
</table>

(Also Includes Window (see page 160) methods)

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart (see page 253)</td>
<td>Gets the Chart view.</td>
</tr>
<tr>
<td>WebBrowser (see page 288)</td>
<td>Gets the Web Browser view.</td>
</tr>
</tbody>
</table>

(Also Includes Window (see page 160) properties)

**VbaViewWindow Examples**

For instructions on setting up VbaView window programs, see "Displaying Data in VbaView Windows" (in the online help).

**Window Object**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 162) ]
To Access

- Instrument.GetWindowByName(Name) (see page 206)
- Windows.Item (see page 266) IndexOrName
- Windows IndexOrName

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoAction (see page 174)</td>
<td>Execute a specific XML-based command action.</td>
</tr>
<tr>
<td>DoCommands (see page 174)</td>
<td>Execute a particular XML-based command.</td>
</tr>
<tr>
<td>Find (see page 184)</td>
<td>Finds a specified data event with optional occurrence and time duration.</td>
</tr>
<tr>
<td>FindNext (see page 188)</td>
<td>Finds the next event by searching forward from the last event found using</td>
</tr>
<tr>
<td></td>
<td>the event specified by the last call to Find (see page 184).</td>
</tr>
<tr>
<td>FindPrev (see page 189)</td>
<td>Finds the previous event by searching backward from the last event found</td>
</tr>
<tr>
<td></td>
<td>using the event specified by the last call to Find (see page 184).</td>
</tr>
<tr>
<td>GoToPosition (see page 212)</td>
<td>Moves the center of the window to a new position.</td>
</tr>
<tr>
<td>QueryCommand (see page 220)</td>
<td>Query for XML-based commands.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusSignals (see page 250)</td>
<td>Gets a collection of the window’s defined bus/signals.</td>
</tr>
<tr>
<td>Name (see page 269)</td>
<td>Gets or sets the name of the window.</td>
</tr>
<tr>
<td>Type (see page 285)</td>
<td>Gets the window’s type.</td>
</tr>
</tbody>
</table>

See Also

- Windows (see page 161) object

Windows Object

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 162) ]

To Access

- Instrument.Windows (see page 289)

Methods

There are no methods.
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum (see page 289)</td>
<td>Used by Visual Basic to support the implementation of For Each ... Next.</td>
</tr>
<tr>
<td>Count (see page 256)</td>
<td>Gets the number of windows in the collection.</td>
</tr>
<tr>
<td>Item (see page 266)</td>
<td>Gets one of the windows in the collection given either an index or name.</td>
</tr>
</tbody>
</table>

### See Also
- Windows (see page 289) property
- Window (see page 160) object
- GetWindowByName (see page 206) method

### Windows Example

#### Visual Basic

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Display all of the window names.
Dim myWindowNames As String
Dim myWindow As AgtLA.Window

For Each myWindow in myInst.Windows
    ' Add the window’s name to the string.
    myWindowNames = myWindowNames + vbNewLine + myWindow.Name
Next
MsgBox "Window names: " + myWindowNames

' Get the compare window using the Windows default Item method, then execute the compare.
myInst.Windows("Compare-1").Execute

#### Visual C++

// This simple Visual C++ Console application demonstrates how to use the Agilent 168x/169x/169xx COM interface to display all the window names and to perform an execute in the Compare window.

// This project was created in Visual C++ Developer. To create a similar project:
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \ 
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
to (search for "TODO" below).
//
#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr))
    {
        try {
            // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            _bstr_t configFile = "C:\LA\Configs\compare.ala";

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            printf("Loading the config file '%s'\n", (char*) configFile);
            pInst->Open(configFile, FALSE, ",", TRUE);

            // Display all of the window names.
            for (long i = 0; i < pWindows->GetCount(); i++)
            {
                AgtLA::IWindowPtr pWindow = pWindows->GetItem(i);
            }
        }
    }
_bstr_t name = pWindow->GetName();
    printf("Window name = %s\n", (char*) name);
}

// Get the Compare window, then execute the compare.
AgtLA::ICompareWindowPtr pCompareWindow = pInst->GetWindowByName("Compare-1");
_bstr_t myCompareOptions = "<Options ReferenceOffset='-2' \n    Range='M1..M2' MaxDifferences='0'/>";
pCompareWindow->PutOptions(myCompareOptions);
pCompareWindow->Execute();
printf("Compare executed using options '%s'\n", (char*) myCompareOptions);
}

} catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
else {
    printf("CoInitialize failed\n");
}

return 0;
}

/////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.
/////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();
    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
else
{
    strcpy(errorStr, desc);
}

printf(" Error Message = %s\n", (char*) errorStr);
Methods

- Add Method (BusSignals Object) (see page 168)
- Add Method (Markers Object) (see page 169)
- AddXML Method (see page 170)
- AddPointArrays Method (see page 170)
- Clear Method (see page 171) (for VbaViewChartData object)
- Clear Method (see page 171) (for VbaViewWebBrowser object)
- ClearOutput Method (see page 172)
- Close Method (see page 172)
- Connect Method (see page 172)
- CopyFile Method (see page 173)
- DeleteFile Method (see page 173)
- DoAction Method (see page 174)
- DoCommands Method (see page 174)
- Draw Method (see page 178)
- Execute Method (see page 178)
- Export Method (see page 179)
- ExportEx Method (see page 180)
- Find Method (see page 184)
- FindNext Method (see page 188)
- FindPrev Method (see page 189)
- GetDataBySample Method (see page 189)
- GetDataByTime Method (see page 192)
- GetGroupCaption Method (see page 193)
- GetLine Method (see page 194)
- GetLineLabel Method (see page 196)
- GetModuleByName Method (see page 196)
- GetNumSamples Method (see page 199)
- GetProbeByName Method (see page 200)
- GetRawData Method (see page 200)
- GetRawTimingZoomData Method (see page 202)
- GetRemoteInfo Method (see page 203)
- GetSampleNumByTime Method (see page 204)
- GetTime Method (see page 204)
• GetToolByNamed Method (see page 205)
• GetValueCaption Method (see page 206)
• GetWindowByName Method (see page 206)
• GoOffline Method (see page 207)
• GoOnline Method (see page 211)
• GoToPosition Method (see page 212)
• Import Method (see page 212)
• ImportEx Method (see page 213)
• InsertLine Method (see page 214)
• IsOnline Method (see page 214)
• IsTimingZoom Method (see page 215)
• New Method (see page 215)
• Open Method (see page 216)
• PanelLock Method (see page 216)
• PanelUnlock Method (see page 220)
• QueryCommand Method (see page 220)
• RecallTriggerByFile Method (see page 223)
• RecallTriggerByName Method (see page 224)
• RecvFile Method (see page 224)
• Remove Method (BusSignals Object) (see page 224)
• Remove Method (Markers Object) (see page 225)
• RemoveAll Method (see page 225)
• RemoveXML Method (see page 225)
• RemoveLine Method (see page 226)
• Reset Method (see page 226)
• Resume Method (see page 227)
• Run Method (Instrument Object) (see page 227)
• Run Method (PattgenModule Object) (see page 228)
• Save Method (see page 228)
• SendFile Method (see page 229)
• SetGroupCaption Method (see page 229)
• SetLine Method (see page 230)
• SetLineLabel Method (see page 230)
• SetValue Method (see page 231)
• SetValueArray Method (see page 231)
- SetValueCaption Method (see page 232)
- SimpleTrigger Method (see page 232)
- Step Method (see page 235)
- Stop Method (Instrument Object) (see page 236)
- Stop Method (PattgenModule Object) (see page 236)
- TestAll Method (see page 237)
- VBADisplayHelpTopic Method (see page 237)
- VBARunMacro Method (see page 237)
- VBARunRPICommand Method (see page 238)
- WaitComplete Method (see page 238)
- WriteOutput Method (see page 242)

Add Method (BusSignals Object)

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To
- BusSignals (see page 113) object

Description
Adds a BusSignal (see page 108) object to the BusSignals (see page 113) collection using specific values.

VB Syntax

```vb
object.Add Name, Channels, [Polarity="+"
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a <strong>BusSignals (see page 113)</strong> object.</td>
</tr>
<tr>
<td>Name</td>
<td>A <strong>String</strong> containing the name of the bus/signal to be added.</td>
</tr>
</tbody>
</table>
Add Method (Markers Object)

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

Applies To
- Markers (see page 123) object

Description
Adds a Marker (see page 123) object to the Markers (see page 123) collection using specific values.

VB Syntax
object.Add Name, TextColor, BackgroundColor, Position

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Markers (see page 123) object.</td>
</tr>
<tr>
<td>Name</td>
<td>A String containing the name of the marker to be added.</td>
</tr>
<tr>
<td>TextColor</td>
<td>A Long representing the color of the text.</td>
</tr>
<tr>
<td>BackgroundColor</td>
<td>A Long representing the color of the background.</td>
</tr>
<tr>
<td>Position</td>
<td>A Double that specifies the position of the marker (in seconds) relative to the trigger.</td>
</tr>
</tbody>
</table>
**AddXML Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

**Applies To**
- Markers (see page 123) object

**Description**
Adds multiple Marker (see page 123) objects to the Markers (see page 123) collection using an XML string.

**VB Syntax**
object.AddXML XMLMarkers

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Markers (see page 123) object.</td>
</tr>
</tbody>
</table>
| XMLMarkers    | An "XML format" (in the online help)String containing the markers to add (see 
"<Markers> element" (in the online help)). |

**AddPointArrays Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

**Applies To**
- VbaViewChartData (see page 157) object

**Description**
Adds an array of points to the chart. This is only valid for chart types AgtChartTypeLine and AgtChartTypeXYScatter.

**VB Syntax**
object.AddPointArrays XValueArray, YValueArray, 
[PointType=AgtDataPointTypeAutomatic], 
[PointSize=AgtDataPointSizeSmall]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
<tr>
<td>XValueArray</td>
<td>An array of Variants that are the X values to set in the chart.</td>
</tr>
<tr>
<td>YValueArray</td>
<td>An array of Variants that are the Y values to set in the chart.</td>
</tr>
<tr>
<td>PointType</td>
<td>An AgtDataPointType enumerated type value that specifies the shape of the point. See the descriptions below.</td>
</tr>
<tr>
<td>PointSize</td>
<td>An AgtDataPointSize enumerated type value that specifies the size of the point. See the descriptions below.</td>
</tr>
</tbody>
</table>

**Remarks**
The **PointType** parameter can have the following values:

<table>
<thead>
<tr>
<th>AgtDataPointType</th>
<th>Enum Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtDataPointTypeNone</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The **PointSize** parameter can have the following values:

<table>
<thead>
<tr>
<th>AgtDataPointType</th>
<th>Enum Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Square</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Diamond</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Triangle</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Invert Triangle</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Star</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Circle</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

### Clear Method (for VbaViewChartData object)

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**Applies To**
- VbaViewChartData (see page 157) object

**Description**
Clears all of the chart data.

**VB Syntax**
```
object.Clear
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
</tbody>
</table>

### Clear Method (for VbaViewWebBrowser object)

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**Applies To**
- VbaViewWebBrowser (see page 159) object

**Description**
Displays an empty web page.

**VB Syntax**
```
object.Clear
```
ClearOutput Method

[Automation Home (see page 3)] [Objects (see page 105)]

Applies To
• VbaViewWindow (see page 160) object

Description
Clears the strings from the output window.

VB Syntax
object.ClearOutput

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWindow (see page 160) object.</td>
</tr>
</tbody>
</table>

Close Method

[Automation Home (see page 3)] [Objects (see page 105)] [Example]

Applies To
• Instrument (see page 121) object

Description
Closes the current configuration.

VB Syntax
object.Close [SaveChanges=False] [SaveFileName=""] [SetupOnly=False]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A Boolean that specifies whether changes to the configuration should be saved.</td>
</tr>
<tr>
<td>SaveFileName</td>
<td>A String that is the name of the file to which the configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A Boolean that specifies whether the configuration file contains captured data or just setup information: True – save only setup information. False – save data and setup information.</td>
</tr>
</tbody>
</table>

Connect Method

[Automation Home (see page 3)] [Objects (see page 105)] [Example]

Requirements

• Version (see page 62): 3.20 or later.
Applies To  •  ConnectSystem (see page 119) object

Description  Connects to the remote logic analyzer system.

VB Syntax  object.Connect [HostNameOrIpAddress=""]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a ConnectSystem (see page 119) object.</td>
</tr>
<tr>
<td>HostNameOrIpAdress</td>
<td>A String that contains the hostname or IP address of the logic analyzer instrument or computer on which the Agilent Logic Analyzer application will run. This parameter is optional.</td>
</tr>
</tbody>
</table>

See Also  •  RecvFile (see page 224) method
•  SendFile (see page 229) method

CopyFile Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  •  Connect (see page 118) object

Description  Copies a file to the instrument file system.

The Instrument (see page 265) property must be called first to establish a connection to the logic analyzer to which the file will be copied.

VB Syntax  object.CopyFile SrcFileName, DestFileName

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Connect (see page 118) object.</td>
</tr>
<tr>
<td>SrcFileName</td>
<td>A String that is the name of the file on the local file system.</td>
</tr>
<tr>
<td>DestFileName</td>
<td>A String that is the name of the file on the instrument file system.</td>
</tr>
</tbody>
</table>

See Also  •  Instrument (see page 265) property

DeleteFile Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  •  Instrument (see page 121) object

Description  Deletes a file on the instrument file system.

VB Syntax  object.DeleteFile FileName
DoAction Method

Applies To
- Instrument (see page 121) object
- Module (see page 128) object
- Probe (see page 135) object
- Tool (see page 152) object
- Window (see page 160) object

Description
Executes a specific command action.

VB Syntax
object.DoAction Action, Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>FileName</td>
<td>A String that is the name of the file on the instrument file system.</td>
</tr>
</tbody>
</table>

Return Value
A Boolean indicating whether the command was successful.

DoCommands Method

Applies To
- Instrument (see page 121) object
- Module (see page 128) object
- Probe (see page 135) object
- Tool (see page 152) object
- Window (see page 160) object

Description
Executes a particular XML-based command.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
<tr>
<td>FileName</td>
<td>A String that is the name of the file on the instrument file system.</td>
</tr>
<tr>
<td>Action</td>
<td>Name of the command to execute. For information about the XML-based actions supported by a tool, see the &quot;Tool Control, COM Automation&quot; topic in the tool's online help.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Command parameters.</td>
</tr>
</tbody>
</table>
VB Syntax  

```vb
object.DoCommands XMLCommand
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list.</td>
</tr>
<tr>
<td>XMLCommand</td>
<td>An XML-format string that configures a module, tool, or window. See &quot;Remarks&quot; below.</td>
</tr>
</tbody>
</table>

**Return Value**  
A **Boolean** indicating whether the command was successful.

**Remarks**  
The XMLCommand format is based on the object type:

<table>
<thead>
<tr>
<th>Object</th>
<th>XMLCommand Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument (see page 121)</td>
<td>A string containing the XML format &quot;&lt;Configuration&gt; element&quot; (in the online help).</td>
</tr>
<tr>
<td>Module (see page 128)</td>
<td>A string containing the XML format &quot;&lt;Module&gt; element&quot; (in the online help) (under Configuration Setup).</td>
</tr>
<tr>
<td>Probe (see page 135)</td>
<td>A string containing the XML format &quot;&lt;Probe&gt; element&quot; (in the online help) (under Configuration Setup).</td>
</tr>
<tr>
<td>Tool (see page 152)</td>
<td>A string containing the XML format &quot;&lt;Tool&gt; element&quot; (in the online help) (under Configuration Setup).</td>
</tr>
<tr>
<td>Window (see page 160)</td>
<td>A string containing the XML format &quot;&lt;Window&gt; element&quot; (in the online help) (under Configuration Setup).</td>
</tr>
</tbody>
</table>

**DoCommands Example**

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Get the MPC8XX Inverse Assembler tool.
Dim myTool As AgtLA.Tool
Set myTool = myInst.GetToolByName("Motorola PowerQUICC (MPC8XX) " + _
"Inverse Assembler-1")

' Query for XML-based command.
Dim XMLCommand As String
If myTool.QueryCommand("GetProperties", XMLCommand) Then
Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to execute XML based
// commands.

// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \ 
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

/////////////////////////////////////////////////////////////////////

// Forward declarations.

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////

// main() entry point.

int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
        }
    }
}
// analysis system hostname.
printf("Connecting to instrument '%s'\n", (char*) hostname);

// Create the connect object and get the instrument object.
AgtLA::IConnectPtr pConnect =
    AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
AgtLA::IInstrumentPtr pInst =
    pConnect->GetInstrument(hostname);

// Load the configuration file.
_bstr_t configFile = "C:\LA\Configs\config.ala";
printf("Loading the config file '%s'\n", (char*) configFile);
pInst->Open(configFile, FALSE, "", TRUE);

// Run the measurement, wait for it to complete.
pInst->Run(FALSE);
pInst->WaitComplete(20);

// Get the MPC8XX Inverse Assembler tool.
_bstr_t myTool =
    "Motorola PowerQUICC (MPC8XX) Inverse Assembler-1";
AgtLA::IToolPtr pTool = pInst->GetToolByName(myTool);

// Query for XML-based command.
BSTR commandXML;
if (pTool->QueryCommand("GetProperties", &commandXML)) {
    printf("MPC8XX IA Properties '%S'\n", (char*) commandXML);
}

// Execute XML based command.
if (pTool->DoCommands(_bstr_t commandXML)) {
    printf("MPC8XX IA DoCommands successful.\n");
}
SysFreeString(commandXML);
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
}
else {
    printf("CoInitialize failed\n");
}
return 0;
}
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();
    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);
        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
            errorStr[nLen - 2] = '\0';
    }
    else
    {
        strcpy(errorStr, desc);
    }
    printf(" Error Message = %s\n", (char*) errorStr);
}

Draw Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To

- VbaViewChart (see page 156) object

Description

Draws the chart.

VB Syntax

object.Draw

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
</tbody>
</table>

Execute Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]

Applies To

- CompareWindow (see page 118) object
**Description**
Executes the compare using the current options.

**VB Syntax**
object.Execute

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a CompareWindow (see page 118) object.</td>
</tr>
</tbody>
</table>

**See Also**
- Differences (see page 261) property
- SampleDifferences (see page 276) property

**Export Method**

```
[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]
```

**Applies To**
- Instrument (see page 121) object

**Description**
Exports data to a file on the instrument file system.

**VB Syntax**
object.Export ExportFileName SourceName [ExportRange=False] [StartRange] [EndRange]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ExportFileName</td>
<td>A String that contains the name of the file (on the instrument file system) to which data is exported.</td>
</tr>
<tr>
<td>SourceName</td>
<td>A String that contains the name of the Module, Tool, or Window whose data will be exported.</td>
</tr>
<tr>
<td>ExportRange</td>
<td>A Boolean that specifies whether a data range is used: True – StartRange and EndRange contain the data range to export. False – exports all the data.</td>
</tr>
<tr>
<td>StartRange EndRange</td>
<td>Variants specifying the data range (see page 180).</td>
</tr>
</tbody>
</table>

**Remarks**
The file is stored onto a drive that is directly accessible by the instrument.

The file format is determined by the ExportFileName suffix. File names with the .csv suffix are either "Standard CSV text file" format or "Pattern Generator CSV text file" format, depending on the SourceName. File names with the .alb suffix are "Module binary file" format (and the SourceName must be a logic analyzer or import module).

The Export method does not support the "Module CSV text file" format. To export this file type, use the ExportEx (see page 180) method.
See Also

- ExportEx (see page 180) method
- Import (see page 212) method
- ImportEx (see page 213) method

Data Ranges

Data ranges are specified by **Variant** start and end parameters.

<table>
<thead>
<tr>
<th>For:</th>
<th>Use the Variant Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample numbers</td>
<td>Integer or Long</td>
</tr>
<tr>
<td>times</td>
<td>Double</td>
</tr>
</tbody>
</table>

In other words:

- To specify a range by sample numbers, use **Integer** or **Long** start and end parameters.
- To specify a range by time, use **Double** start and end parameters.

See Also

- Export (see page 179) method
- GetNumSamples (see page 199) method
- GetTime (see page 204) method

**ExportEx Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To

- **Instrument** (see page 121) object

Description

Exports data to a file on the instrument file system.

**VB Syntax**

```
object.ExportEx ExportFileName SourceName [ExportRange=False] [StartRange] [EndRange] [FileType=""] [FileOptions=""]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ExportFileName</td>
<td>A <strong>String</strong> that contains the name of the file (on the instrument file system) to which data is exported.</td>
</tr>
</tbody>
</table>
### Parameters | Definition
--- | ---
SourceName | A **String** that contains the name of the Module, Tool, or Window whose data will be exported. When the "Module CSV text file" or "Module binary file" File Type is specified, the SourceName must be a logic analyzer or import module. Timing zoom data can be exported separately using the SourceName syntax of "<module_name>:TimingZoom", for example, "My 16950A-1:TimingZoom". When the "Pattern Generator CSV text file" File Type is specified, the SourceName must be a pattern generator module.

ExportRange | A **Boolean** that specifies whether a data range is used: **True** – StartRange and EndRange contain the data range to export. **False** – exports all the data.

StartRange EndRange | **Variants** specifying the data range (see page 180).
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SourceName</strong></td>
<td>A <strong>String</strong> that contains the name of the Module, Tool, or Window whose data will be exported. When the &quot;Module CSV text file&quot; or &quot;Module binary file&quot; File Type is specified, the SourceName must be a logic analyzer or import module. Timing zoom data can be exported separately using the SourceName syntax of &quot;&lt;module_name&gt;:TimingZoom&quot;, for example, &quot;My 16950A-1:TimingZoom&quot;. When the &quot;Pattern Generator CSV text file&quot; File Type is specified, the SourceName must be a pattern generator module.</td>
</tr>
<tr>
<td><strong>ExportRange</strong></td>
<td>A <strong>Boolean</strong> that specifies whether a data range is used: <strong>True</strong> – StartRange and EndRange contain the data range to export. <strong>False</strong> – exports all the data.</td>
</tr>
<tr>
<td><strong>StartRange</strong></td>
<td><strong>Variants</strong> specifying the data range (see page 180).</td>
</tr>
<tr>
<td><strong>EndRange</strong></td>
<td><strong>Variants</strong> specifying the data range (see page 180).</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FileType</strong></td>
<td>A <code>String</code> that identifies the type of data you want to export. This is</td>
</tr>
<tr>
<td></td>
<td>the same string that you see in the <em>Agilent Logic Analyzer</em> application's</td>
</tr>
<tr>
<td></td>
<td>Export dialog:</td>
</tr>
<tr>
<td></td>
<td>- &quot;Standard CSV text file&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;Module CSV text file&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;Module binary file&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;Pattern Generator CSV text file&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;16700 ASCII File Format&quot;</td>
</tr>
<tr>
<td><strong>FileOptions</strong></td>
<td>An XML format <code>String</code> that lets you specify export options (as you can</td>
</tr>
<tr>
<td></td>
<td>with the <em>Agilent Logic Analyzer</em> application’s Export dialog Options...</td>
</tr>
<tr>
<td></td>
<td>button). For example:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Options</code></td>
</tr>
<tr>
<td></td>
<td>SeparationCharacters=','</td>
</tr>
<tr>
<td></td>
<td>WriteLineNumberColumn='T'</td>
</tr>
<tr>
<td></td>
<td>LineNumberColumnName='Line Number'</td>
</tr>
<tr>
<td></td>
<td>IncludeHeader='T'</td>
</tr>
<tr>
<td></td>
<td>WriteFixedWidthColumns='F'</td>
</tr>
<tr>
<td></td>
<td><code>/&gt;</code></td>
</tr>
</tbody>
</table>

The `<Options>` element attribute values can be:

- `SeparationCharacters` — `string`
- `WriteLineNumberColumn` — 'F' (false) or 'T' (true)
- `LineNumberColumnName` — `string`
- `IncludeHeader` — 'F' (false) or 'T' (true)
- `WriteFixedWidthColumns` — 'F' (false) or 'T' (true)

Certain file types support certain file options (as in the *Agilent Logic Analyzer* application):

- "Standard CSV text file" — `SeparationCharacters`, `WriteLineNumberColumns`, and `LineNumberColumnName`.
- "Module CSV text file" — `SeparationCharacters`, `WriteLineNumbers`, `LineNumberColumnName`, and `IncludeHeader`.
- "Module binary file" — `IncludeHeader`.
- "Pattern Generator CSV text file" — `SeparationCharacters`.
- "16700 ASCII File Format" — (none).

### Remarks

The file is stored onto a drive that is directly accessible by the instrument.

### See Also

- Export (see page 179) method
- Import (see page 212) method
- ImportEx (see page 213) method
## Find Method

Applies To  
- **Window (see page 160) object**

Description  
Finds a specified data event with optional occurrence and time duration.

VB Syntax  
```vb
object.Find Event, [Occurrence=1], [Direction="F"], [From="Display Center"], [When="Present"], [Duration=""]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Window (see page 160) object.</td>
</tr>
<tr>
<td>Event</td>
<td>A <strong>String</strong> containing the event to find. The event can be a simple string (see Event (see page 233)) or, for more complex searches, an XML format string (see &quot;&lt;Event&gt; element (under Find)&quot; (in the online help)).</td>
</tr>
<tr>
<td>Occurrence</td>
<td>A <strong>Long</strong> containing the number of occurrences of the Event parameter.</td>
</tr>
<tr>
<td>Direction</td>
<td>A <strong>String</strong> containing the direction to search. &quot;F&quot; searches forward, &quot;B&quot; searches backward.</td>
</tr>
</tbody>
</table>
| From       | A **String** containing the position to start searching. This can be any of the following:  
- "Display Center"  
- "Beginning Of Data"  
- "End Of Data"  
- "Trigger"  
- Name of a currently defined marker.  
Note that these strings are case-sensitive.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
</table>
| When | A **String** specifying a time duration or other operator.  
- "Present"
- "Not Present"
- "Present>" (Duration must contain only one time value)
- "Present>=" (Duration must contain only one time value)
- "Present<" (Duration must contain only one time value)
- "Present<=" (Duration must contain only one time value)
- "Present for Range" (Duration must contain a time range)
- "Not In Range" (Duration must contain a time range)
- "Entering"
- "Exiting"
- "Transitioning"
Note that these strings are case-sensitive. |
| Duration | A **String** containing a time duration which is only valid for certain 'When' qualifiers above. The format of this string can be either a time value or time range value. A time range contains a time value followed by the string ".." followed by another time value.
A time value contains a number followed by a time unit.
A time unit can be any of the strings:
- "ps" - picoseconds
- "ns" - nanoseconds
- "us" - microseconds
- "ms" - milliseconds
- "s" - seconds
- "Gs" - gigaseconds
Examples:
- "1 ps"
- "1ns..30ns" |

**Return Value**
A **FindResult** (see page 119) object containing the results of the find.

**See Also**
- FindNext (see page 188) method
- FindPrev (see page 189) method
- FindResult (see page 119) object

**Find Example**
**Visual Basic**
```vbnet
Dim myWindow As Window
Set myWindow = myInst.Windows("Listing-1")

' Find using a simple event.
Dim myResult As FindResult
Set myResult = myWindow.Find("My Bus 1 = h55", 1, "F", _
    "Beginning Of Data")
If myResult.Found Then
    ' The event was found...
```
End If

' Find the same event using XML.
Dim myXMLEvent As String
myXMLEvent = "<Event>" + _
"<BusSignal Name='My Bus 1' Operator='=' " + _
"Value='h55' />" + _
"</Event>"
Set myResult = myWindow.Find(myXMLEvent, 1, "F", "Beginning Of Data")
If myResult.Found Then
   ' The event was found...
End If

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to find a specified data
// event.

// This project was created in Visual C++ Developer. To create a
// similar project:
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).

#include "stdafx.h"

////////////////////////////////////////////////////////////////////
// Forward declarations.
////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& err);

////////////////////////////////////////////////////////////////////
// main() entry point.
////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
   printf("*** Main()\n");

   // Initialize the Microsoft COM/ActiveX library.
   HRESULT hr = CoInitialize(0);
if (SUCCEEDED(hr))
{
    try {
        // Catch any unexpected run-time errors.
        _bstr_t hostname = "mtx33"; // TODO, use your logic
        // analysis system hostname.
        printf("Connecting to instrument '%s\n", (char*) hostname);

        // Create the connect object and get the instrument object.
        AgtLA::IConnectPtr pConnect =
            AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
        AgtLA::IInstrumentPtr pInst =
            pConnect->GetInstrument(hostname);

        // Run the measurement, wait for it to complete.
        pInst->Run(FALSE);
        pInst->WaitComplete(20);

        // Get a specific window.
        _bstr_t myListing = "Listing-1";
        AgtLA::IWindowPtr pWindow = pInst->GetWindowByName(myListing);

        // Find using a simple event.
        _bstr_t myEvent = "My Bus 1 = h55";
        AgtLA::IFindResultPtr pFindResult = pWindow->Find(myEvent, 1,
            "F", "Beginning Of Data", "Present", "");
        if (pFindResult->GetFound()) {
            _bstr_t myTimeFound = pFindResult->GetTimeFoundString();
            printf("Event '%s' was found at '%s'.\n", (char*) myEvent,
                (char *) myTimeFound);
        } else {
            printf("Event '%s' was not found.\n", (char*) myEvent);
        }

        // Find the same event using XML.
        _bstr_t myXMLEvent = "<Event><BusSignal Name='My Bus 1' \"
            Operator='=' Value='h55' /></Event>";
        pFindResult = pWindow->Find(myXMLEvent, 1, "F", 
            "Beginning Of Data", "Present", "");
        if (pFindResult->GetFound()) {
            _bstr_t myTimeFound = pFindResult->GetTimeFoundString();
            printf("XML event '%s' was found at '%s'.\n", (char*) myXMLEvent,
                (char *) myTimeFound);
        } else {
            printf("XML event '%s' was not found.\n", (char*) myXMLEvent);
        }
    }
    catch (_com_error& e) {
        DisplayError(e);
    }

    // Uninitialize the Microsoft COM/ActiveX library.
    CoUninitialize();
}
else
{
  printf("CoInitialize failed\n");
}

return 0;
}

/////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.
//
// void DisplayError(_com_error& error)
{
  printf("*** DisplayError()\n");

  printf("Fatal Unexpected Error:\n");
  printf("  Error Number = %08lx\n", error.Error());

  static char errorStr[1024];
  _bstr_t desc = error.Description();

  if (desc.length() == 0)
  {
    // Don't have a description string.
    strcpy(errorStr, error.ErrorMessage());
    int nLen = lstrlen(errorStr);

    // Remove funny carriage return ctrl<M>.
    if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
    {
      errorStr[nLen - 2] = '\0';
    }
  }
  else
  {
    strcpy(errorStr, desc);
  }

  printf("  Error Message = %s\n", (char*) errorStr);
}

FindNext Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 185) ]

Applies To

• Window (see page 160) object

Description
Finds the next event by searching forward from the last event found using the event specified by the last call to Find (see page 184).

VB Syntax
object.FindNext
### FindPrev Method

**Applies To**  
- Window (see page 160) object

**Description**  
Finds the previous event by searching backward from the last event found using the event specified by the last call to Find (see page 184).

**VB Syntax**  
object.FindPrev

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Window (see page 160) object.</td>
</tr>
</tbody>
</table>

**Return Value**  
A FindResult (see page 119) object containing the results of the find.

**See Also**  
- FindResult (see page 119) object
- Find (see page 184) method
- FindPrev (see page 189) method

### GetDataBySample Method

**Applies To**  
- SampleBusSignalData (see page 138) object

**Description**  
Given a range, returns an array of acquired data. **GetDataBySample** returns the data within a trigger relative sample range.

**Parameters Definition**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Window (see page 160) object.</td>
</tr>
</tbody>
</table>

**Return Value**  
A FindResult (see page 119) object containing the results of the find.

**See Also**  
- FindResult (see page 119) object
- Find (see page 184) method
- FindNext (see page 188) method

---

**NOTE**  
The data can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s *WaitComplete* (see page 238) method to make sure the hardware is stopped.
VB Syntax  
object.GetDataBySample StartSample, EndSample, DataType, NumRowsRet

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData (see page 138) object.</td>
</tr>
<tr>
<td>StartSample</td>
<td>A Long containing the first sample to upload.</td>
</tr>
<tr>
<td>EndSample</td>
<td>A Long containing the last sample to upload. EndSample must be greater than or equal to StartSample.</td>
</tr>
<tr>
<td>DataType</td>
<td>Specifies the type of data to return. See DataTypes and Return Values (see page 190).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumRowsRet</td>
<td>A Long initialized by this method to the number of rows being returned in the array.</td>
</tr>
</tbody>
</table>

**Return Values**  
See DataTypes and Return Values (see page 190).

**See Also**  
- GetDataByTime (see page 192) method
- StartSample (see page 279) property
- EndSample (see page 261) property
- GetTime (see page 204) method

**DataTypes and Return Values**

<table>
<thead>
<tr>
<th>AgtDataType</th>
<th>Enum Value</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtDataRaw</td>
<td>&amp;H0001 (1)</td>
<td>Returns an array of Bytes. The total size of the array is NumRowsRet multiplied by the value in the BusSignal (see page 108) object’s ByteSize (see page 251) property. This can hold the maximum BusSignal size of 128 bits of unsigned data. This type is the most efficient in terms of bytes transferred. Using this type, only the smallest number of bytes needed to represent every channel in a bus/signal will be transferred. The bus/signal values are stored in the array from MSB to LSB.</td>
</tr>
<tr>
<td>AgtDataDecimal</td>
<td>&amp;H0002 (2)</td>
<td>Returns an array of <strong>Variants</strong>. The <strong>Variant</strong> contains a decimal data type that holds 96 bits of unsigned and signed integer data. Decimals are stored as 96-bit unsigned integers scaled by a variable power of 10. The power of 10 scaling factor specifies the number of digits to the right of the decimal point and ranges from 0 to 28. This data type can be used when the <strong>BusSignalType</strong> (see page 249) property is <strong>AgtBusSignalSampleNum</strong>, when the <strong>GetTime</strong> (see page 204) method is called, when the bus/signal is oscilloscope voltage data, or when the bus/signal you are getting data for is less than 96 bits wide, unsigned. No error is returned if the data is truncated.</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AgtDataLong</td>
<td>&amp;H0003 (3)</td>
<td>Returns an array of <strong>Longs</strong>. This holds 31 bits of unsigned integer data and 32 bits of signed integer data. This data type can be used when the <strong>BusSignalType</strong> (see page 249) property is <strong>AgtBusSignalSampleNum</strong> or when the bus/signal you are getting data for is less than 32 bits wide, unsigned. No error is returned if the data is truncated.</td>
</tr>
<tr>
<td>AgtDataTime</td>
<td>&amp;H0004 (4)</td>
<td>Returns an array of <strong>Doubles</strong>. This data type is only valid when the <strong>GetTime</strong> (see page 204) method is called. You can also use <strong>AgtDataDouble</strong> and <strong>AgtDataDecimal</strong> to access time values as well.</td>
</tr>
<tr>
<td>AgtDataStringDec</td>
<td>&amp;H0005 (5)</td>
<td>Returns an array of <strong>Strings</strong>, formatted in decimal.</td>
</tr>
<tr>
<td>AgtDataStringHex</td>
<td>&amp;H0006 (6)</td>
<td>Returns an array of <strong>Strings</strong>, formatted in hexadecimal. When the <strong>GetTime</strong> (see page 204) method is called, the value is formatted as a hex string in units of $10^{-24}$ seconds. When the bus/signal is oscilloscope voltage data, the value is formatted as a hex string in units of $10^{-12}$ volts. In both cases, the string is an exact representation of the internal value; therefore, no information is lost.</td>
</tr>
</tbody>
</table>
# GetDataByTime Method

<table>
<thead>
<tr>
<th>OR’ed in AgtDataType</th>
<th>Enum Value</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtDataSubrows</td>
<td>&amp;H0100 (256)</td>
<td>Returns an array of arrays of type specified by the lower 8 bits of <code>EnumValue</code>. For example, if the <code>EnumValue</code> is a bitwise OR of AgtDataString and AgtDataSubrows (see code examples below), an array of string arrays will be returned. This is used to return multiple rows of data per sample, for example, when an inverse assembler returns multiple rows of decoded strings per sample. If the sample does not contain subrows, the array for that sample will typically contain one value; however, when the BusSignalType (see page 249) property is <code>AgtBusSignalGenerated</code>, there are cases when the array may be empty.</td>
</tr>
<tr>
<td>AgtDataSigned</td>
<td>&amp;H0200 (512)</td>
<td>Returns signed values of type specified by the lower 8 bits of <code>EnumValue</code>. This value is ignored for <code>EnumValueAgtDataRaw</code>. For example, if the <code>EnumValue</code> is a bitwise OR of AgtDataDouble and AgtDataSigned, each sample will be converted to a signed value and returned as an array of doubles.</td>
</tr>
</tbody>
</table>

Returns an array of Strings using the default format.

Returns an array of Doubles. This holds 52 bits of unsigned integer data and 53 bits of signed integer data. This data type can be used when the BusSignalType (see page 249) property is `AgtBusSignalSampleNum`, when the GetTime (see page 204) method is called for any bus/signal, when the bus/signal is oscilloscope voltage data, or when the bus/signal you are getting data for is less than 52 bits wide, unsigned. No error is returned if the data is truncated.
Applies To  •  SampleBusSignalData (see page 138) object

Description  Given a range, returns an array of acquired data. GetDataByTime returns the data within a trigger relative time range.

NOTE  The data can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s WaitComplete (see page 238) method to make sure the hardware is stopped.

VB Syntax  object.GetDataByTime StartTime, EndTime, DataType, NumRowsRet

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData (see page 138) object.</td>
</tr>
<tr>
<td>StartTime</td>
<td>A Double containing the starting time (in seconds) to upload. Double values can be expressed as mmmEeee or mmmDeee, in which mmm is the mantissa and eee is the exponent (a power of 10); for example, a StartTime value of 450E-9 is 450 nanoseconds.</td>
</tr>
<tr>
<td>EndTime</td>
<td>A Double containing the ending time (in seconds) to upload. EndTime must be greater than or equal to StartTime.</td>
</tr>
<tr>
<td>DataType</td>
<td>Specifies the type of data to return. See DataTypes and Return Values (see page 190).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumRowsRet</td>
<td>A Long initialized by this method to the number of rows being returned in the array.</td>
</tr>
</tbody>
</table>

Return Values  See DataTypes and Return Values (see page 190).

See Also  •  GetDataBySample (see page 189) method
          •  StartTime (see page 279) property
          •  EndTime (see page 262) property
          •  GetTime (see page 204) method

GetGroupCaption Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To  •  VbaViewChartData (see page 157) object

Description  Gets the caption associated with a group (row).

VB Syntax  object.GetGroupCaption Group
### GetLine Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 131) ]

**Applies To**
- PattgenModule (see page 129) object

**Description**
Gets an instruction or vector at line number.

**VB Syntax**
object.GetLine LineNumber

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>A Long representing the line number from which to get the instruction or vector.</td>
</tr>
</tbody>
</table>

**Return Value**
A **String** containing the InstructionOrVector (see page 194) at the specified line number.

**See Also**
- SetLine (see page 230) method

### InstructionOrVector

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 131) ]

**Applies To**
- GetLine (see page 194) method
- InsertLine (see page 214) method
- SetLine (see page 230) method

**Description**
An **InstructionOrVector** string contains a pattern generator instruction or vector.

**Syntax**
The syntax is like the syntax used in the user-interface with the exception of Vectors. With vectors, a "label=value" syntax is used to lessen ambiguity. The syntax is as follows:
- Vector <label_value>, <label_value>, ...
- Start Loop Repeat <integer> [ times | time ]
- User-Defined Macro '<macro_name>' (<parameter_value>, <parameter_value>, ...)
- Wait for External Event [ A | B | C | D ] { = ([ <wait_pattern> + <wait_pattern> + ... | Any | None ] ) }
- Wait for Arm in from [ '<module_name>' | 'External Trigger' ] [ , '<module_name>' | 'External Trigger' ]*
- Send Arm out to [ '<module_name>' | 'External Trigger' ] [ , '<module_name>' | 'External Trigger' ]*
- Break

Where:

<table>
<thead>
<tr>
<th>&lt;label_value&gt;</th>
<th>'&lt;label_name&gt;' = &lt;value&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;parameter_value&gt;</td>
<td>'&lt;parameter_name&gt;' = &lt;value&gt;</td>
</tr>
<tr>
<td>&lt;wait_pattern&gt;</td>
<td>A 3-digit binary number as in the user-interface (for example, 001).</td>
</tr>
<tr>
<td>&lt;module_name&gt;</td>
<td>The name of a module in the frame.</td>
</tr>
</tbody>
</table>
| <value> | Values are formatted consistently with XML and COM number formatting. Numbers are case-insensitive and must contain a number base prefix:
  - h — hexadecimal
  - o — octal
  - b — binary
  - d — decimal
Numbers can optionally contain don’t care symbols:
  - X — don’t care
Some example values are:
  - hfx
  - o72
  - b11110000
  - d24
  - b1111xxxx

Unsupported Instructions

The following instructions are not supported for inserting or modifying because they always exist within the main sequence or are implicitly created:

- Init Start
- Init End
- Main Start
Examples

- Vector 'My Bus 1' = h1, 'My Bus 2' = h2, 'My Bus 3' = h3, 'My Bus 4' = h4, 'My Bus 5' = h5

- Start Loop Repeat 25 times

- User-Defined Macro 'My Macro 1' ("MyParam1"=11, "MyParam2"=22, "MyParam3"=33)

- Wait for External Event A = (000 + 001 + 010)

- Wait for Arm in from 'External Trigger', 'My 1674x/5x-1'

- Send Arm out to 'My 1674x/5x-1'

- Break

See Also

- GetLine (see page 194) method
- InsertLine (see page 214) method
- SetLine (see page 230) method

GetLineLabel Method

Description

Gets a vector's label value at line number.

**VB Syntax**

```
object.GetLineLabel Linenumber, LabelName
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>A Long representing the line number from which to get the instruction or vector.</td>
</tr>
<tr>
<td>LabelName</td>
<td>A String name of label whose value you wish to get.</td>
</tr>
</tbody>
</table>

Return Value

A String containing the value of the label name.

See Also

- SetLineLabel (see page 230) method

GetModuleByName Method

Parameters Definition

- **object**
  - An expression that evaluates to a PattgenModule (see page 129) object.
- **LineNumber**
  - A Long representing the line number from which to get the instruction or vector.
- **LabelName**
  - A String name of label whose value you wish to get.

Return Value

A String containing the value of the label name.

See Also

- SetLineLabel (see page 230) method
Applies To

- **Instrument** (see page 121) object

Description

Given a module name, returns its corresponding hardware module object.

VB Syntax

```vbnet
object.GetModuleByName Name
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Name</td>
<td>A String containing the module’s name as defined by the Name (see page 269) property.</td>
</tr>
</tbody>
</table>

Return Value

A hardware Module (see page 128) object with the module name given.

See Also

- Modules (see page 268) property
- Name (see page 269) property

GetModuleByName Example

**Visual Basic**

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.

' When "using the Advanced Customization Environment (ACE)" (in the online help),
' the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Get the module named "My 16910A-1".
Dim myModule As AgtLA.Module
Set myModule = myInst.GetModuleByName("My 16910A-1")

' Display the module status.
MsgBox "Module: " + myModule.Name + ", status: " + myModule.Status

**Visual C++**

```c++
// This simple Visual C++ Console application demonstrates how to use the Agilent 168x/169x/169xx COM interface to get a module by name and display its status.

// This project was created in Visual C++ Developer. To create a similar project:

// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path
```
```cpp
// instdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"

// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr))
    {
      try { // Catch any unexpected run-time errors.
        _bstr_t hostname = "mtx33"; // TODO, use your logic

        printf("Connecting to instrument '%s'\n", (char*) hostname);

        // Create the connect object and get the instrument object.
        AgtLA::IConnectPtr pConnect = 
            AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
        AgtLA::IInstrumentPtr pInst = 
            pConnect->GetInstrument(hostname);

        // Get a specific analyzer module.
        _bstr_t moduleName = "My 16910A-1";
        AgtLA::IAnalyzerModulePtr pAnalyzer = 
            pInst->GetModuleByModuleName(moduleName);

        // Display the module status.
        _bstr_t name = pAnalyzer->GetName();
        _bstr_t status = pAnalyzer->GetStatus();
        printf("Module '%s', status '%s'.\n", (char*) name, (char*) status);
      } catch (_com_error& e) {
        DisplayError(e);
    }
```
// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
}
else
{
    printf("CoInitialize failed\n");
}
return 0;

////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception
// information.
////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error: \n");
    printf(" Error Number = %08lx\n", error.Error());
    static char errorStr[1024];
    _bstr_t desc = error.Description();
    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);
        // Remove funny carriage return ctrl<\M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\'0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }
    printf(" Error Message = %s\n", (char*) errorStr);
}

GetNumSamples Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 139) ]

Applies To  • SampleBusSignalData (see page 138) object
Description
Given a range, returns the number of samples stored.

NOTE
The data can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s WaitComplete (see page 238) method to make sure the hardware is stopped.

VB Syntax
object.GetNumSamples [StartPosition], [EndPosition]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData (see page 138) object.</td>
</tr>
<tr>
<td>StartPosition</td>
<td>Variants specifying the data range (see page 180). EndPosition</td>
</tr>
<tr>
<td>EndPosition</td>
<td>must be greater than or equal to StartPosition.</td>
</tr>
</tbody>
</table>

Return Values
A Long containing the number of samples in the range.

GetProbeByName Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To
• Instrument (see page 121) object

Description
Given a probe name, returns the corresponding probe object.

VB Syntax
object.GetProbeByName Name

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Name</td>
<td>A String containing the probe’s name as defined by the Name (see page 269) property.</td>
</tr>
</tbody>
</table>

Return Value
A Probe (see page 135) object with the name given.

See Also
• Probe (see page 135) object
• Probes (see page 135) object
• Instrument (see page 121) object
• Name (see page 269) property

GetRawData Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]
Applies To
• AnalyzerModule (see page 106) object

Description
Given a range, returns the raw analyzer data.

NOTE
The data can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s WaitComplete (see page 238) method to make sure the hardware is stopped.

VB Syntax
object.GetRawData StartPosition, EndPosition, NumBytesPerRow, NumRowsRet

Parameters | Definition
--- | ---
object | An expression that evaluates to an AnalyzerModule (see page 106) object.
StartPosition EndPosition | Variants specifying the data range (see page 180). EndPosition must be greater than or equal to StartPosition.

Returns | Definition
--- | ---
NumBytesPerRow | A Long initialized by this method to the width of each sample row being returned in the array.
NumRowsRet | A Long initialized by this method to the number of rows being returned in the array.

Return Values
Returns a Variant array of Bytes. The total size of the array is NumRowsRet multiplied by NumBytesPerRow.

The data is returned in the following format (which is reversed from the byte order that appears in the Buses/Signals Setup dialog):

Pod 1, Pod 2, Pod 3, ..., Clock

For example, if the logic analyzer has two pods:
The data would be stored in array like:

```
Pod 1    Pod 1    Pod 2    Pod 2    Clocks
7..0     15..8   7..0     15..8   1..0
        -------- -------- -------- -------- --------
```

The Clock bytes are rounded up to the nearest byte; for example, if there are 10 clock channels, the data is stored in two bytes.

See Also
- GetRawTimingZoomData (see page 202) method

**GetRawTimingZoomData Method**

VB Syntax
```vbnet
object.GetRawTimingZoomData StartPosition, EndPosition, NumBytesPerRow, NumRowsRet
```

NOTE
The data can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s **WaitComplete** (see page 238) method to make sure the hardware is stopped.
Return Values

Returns an array of **Bytes**. The total size of the array is NumRowsRet multiplied by NumBytesPerRow.

The data is returned in the following format (which is reversed from the bit order that appears in the Buses/Signals Setup dialog):

Pod 1, Pod 2, Pod 3, ..., Clock

The Clock bytes are rounded up to the nearest byte; for example, if there are 10 clock channels, the data is stored in two bytes.

**See Also**
- GetRawData (see page 200) method

### GetRemoteInfo Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- Connect (see page 118) object

**Description**

Gets the logic analyzer's remote user login and computer name.

This method gives you information about who is remotely connected via COM to the logic analyzer. This is a passive interrogation and does not modify the current values returned by RemoteComputerName (see page 274) and RemoteUserName (see page 275).

**VB Syntax**

```vbnet
object.GetRemoteInfo HostNameOrIPAddress, RemoteUserName, RemoteComputerName
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Connect (see page 118) object.</td>
</tr>
<tr>
<td>StartPosition EndPosition</td>
<td><strong>Variants</strong> specifying the data range (see page 180). <strong>EndPosition</strong> must be greater than or equal to <strong>StartPosition</strong>.</td>
</tr>
<tr>
<td>NumBytesPerRow</td>
<td><strong>Long</strong> initialized by this method to the width of each sample row being returned in the array.</td>
</tr>
<tr>
<td>NumRowsRet</td>
<td><strong>Long</strong> initialized by this method to the number of rows being returned in the array.</td>
</tr>
</tbody>
</table>
### GetSampleNumByTime Method

**Description**

Gets the closest sample number corresponding to the time given.

**NOTE**

The sample number can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s `WaitComplete` method to make sure the hardware is stopped.

**VB Syntax**

```vbscript
object.GetSampleNumByTime Time
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData object.</td>
</tr>
<tr>
<td>Time</td>
<td>A <code>Double</code> containing the time that you want to get the sample number for. Double values can be expressed as <code>mmmEeee</code> or <code>mmmDeee</code>, in which <code>mmm</code> is the mantissa and <code>eee</code> is the exponent (a power of 10); for example, a Time value of 450E-9 is 450 nanoseconds.</td>
</tr>
</tbody>
</table>

**Return Values**

A `Long` containing the sample number.

### GetTime Method

**Description**

Given a range, returns the time for this bus/signal in the format specified by the data type given. `GetTime` gets the time values for the specific bus/signal.
The time can only be returned when the hardware is stopped. Before calling this method, call the Instrument object’s `WaitComplete` (see page 238) method to make sure the hardware is stopped.

### VB Syntax

```vb
object.GetTime StartPosition, EndPosition, DataType, NumRowsRet
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a SampleBusSignalData (see page 138) object.</td>
</tr>
<tr>
<td>StartPosition, EndPosition</td>
<td>Variants specifying the data range (see page 180). <code>EndPosition</code> must be greater than or equal to <code>StartPosition</code>.</td>
</tr>
<tr>
<td>DataType</td>
<td>Specifies the type of data to return. See DataTypes and Return Values (see page 190).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumRowsRet</td>
<td>A Variant initialized by this method to the number of rows being returned in the time array.</td>
</tr>
</tbody>
</table>

**Return Values**

See DataTypes and Return Values (see page 190).

**See Also**

- GetDataByTime (see page 192) method
- GetDataBySample (see page 189) method

### GetToolByName Method

```vb
[Automation Home (see page 3)] [Objects (see page 105)] [Example]```

- **Applies To**
  - Instrument (see page 121) object
- **Description**
  Given a tool name, returns its corresponding tool object.
- **VB Syntax**
  ```vb
  object.GetToolByName Name
  ```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Name</td>
<td>A String containing the tool’s name as defined by the Name (see page 269) property.</td>
</tr>
</tbody>
</table>

**Return Value**

A Tool (see page 152) object with the tool name given.

**See Also**

- Tools (see page 284) property
GetValueCaption Method

A VbaViewChartData (see page 157) object

Description
Gets the caption associated with all values at index (column).

VB Syntax
object.GetValueCaption Index

Return Value
A String containing the value caption.

See Also
SetValueCaption (see page 232) method

GetWindowByName Method

An Instrument (see page 121) object

Description
Given a window name, returns the corresponding window object.

VB Syntax
object.GetWindowByName Name

Return Value
A Window (see page 160) object with the tool name given.

See Also
Window (see page 160) object
Windows (see page 161) object
Instrument (see page 121) object
Name (see page 269) property
GoOffline Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 207) ]

Applies To
- Instrument (see page 121) object

Description
Disconnects the user interface from the logic analyzer frame.

VB Syntax
object.GoOffline

### Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

See Also
- GoOnline (see page 211) method
- IsOnline (see page 214) method

GoOffline Example

Visual Basic
Option Explicit ' Must define all variables.

Sub Main()

' Define the logic analysis systems being used.
Dim myFirstLAS As String
Dim mySecondLAS As String
myFirstLAS = "mtx33"
mySecondLAS = "col-mil20"

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it
to access the Instrument object. In this example, "myInst" represents the Instrument object.

Dim myConnect As AgtLA.Connect
Dim myInst As AgtLA.Instrument
Set myConnect = CreateObject("AgtLA.Connect")
Set myInst = myConnect.Instrument(myFirstLAS)

' When "using the Advanced Customization Environment (ACE)" (in the online help),
' the Instrument object is already created and is globally accessible using "AgtLA". In this example, comment out
' the preceding four lines; then, substitute "myInst" with
' "AgtLA" to access the global Instrument object in VBA.

' Display whether the instrument is offline or online.
DisplayConnected myInst, myFirstLAS

' Go offline.
myInst.GoOffline
DisplayConnected myInst, myFirstLAS

' Go online to second logic analysis system.
myInst.GoOnline (mySecondLAS)
DisplayConnected myInst, myFirstLAS

' Go offline.
myInst.GoOffline
DisplayConnected myInst, myFirstLAS

' Go online to first logic analysis system.
myInst.GoOnline (myFirstLAS)
DisplayConnected myInst, myFirstLAS

End Sub

Private Sub DisplayConnected(inst As AgtLA.Instrument, system As String)
    Dim connectedTo As String
    If inst.IsOnline(connectedTo) Then
        MsgBox "LA system: " + system + ", is online, connected to: " + connectedTo
    Else
        MsgBox "LA system: " + system + ", is offline."
    End If
End Sub

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to take a logic analysis
// system offline and then go back online again.

// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)

// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \n// COM Automation/agClientSvr.dll"

// To run, you need to specify the logic analysis systems to connect
// to (search for "TODO" below).

#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
//
// void DisplayConnected(
//    AgtLA::IInstrumentPtr pInst,
//    _bstr_t system);

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
//
// main() entry point.
//
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);
    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t myFirstLAS = "mtx33"; // TODO, use your first logic
            // analysis system hostname.
            _bstr_t mySecondLAS = "col-mil20"; // TODO, use your second
            // logic analysis system
            // hostname.
            printf("Connecting to instrument '%s'\n", (char*) myFirstLAS);

            // Create the connect object and get the instrument object.
            AgtLA::IConnectPtr pConnect =
                AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
            AgtLA::IInstrumentPtr pInst =
                pConnect->GetInstrument(myFirstLAS);

            // Display whether the instrument is offline or online.
            DisplayConnected(pInst, myFirstLAS);

            // Go offline.
            pInst->GoOffline();
            DisplayConnected(pInst, myFirstLAS);

            // Go online to second logic analysis system.
            pInst->GoOnline(mySecondLAS, 1, FALSE, "", FALSE);
            DisplayConnected(pInst, myFirstLAS);

            // Go offline.
            pInst->GoOffline();
            DisplayConnected(pInst, myFirstLAS);

            // Go online to first logic analysis system.
            pInst->GoOnline(myFirstLAS, 1, FALSE, "", FALSE);
            DisplayConnected(pInst, myFirstLAS);
        }
    }
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
} else {
    printf("CoInitialize failed\n");
}
return 0;
}

/////////////////////////////////////////////////////////////////////
// Displays whether a logic analysis system if offline or online,  
// and if online the system it it connected to.                  
/////////////////////////////////////////////////////////////////////
void DisplayConnected( 
    AgtLA::IInstrumentPtr pInst,  
    _bstr_t system)  
{
    printf("*** DisplayConnected()\n");
    BSTR connectedTo;
    if (pInst->IsOnline(&connectedTo)) {
        printf("LA system '%s', is online, connected to '%s'.\n", 
            (char*) system, (char*) connectedTo);
    } else {
        printf("LA system '%s', is offline.\n", (char*) system);
    }
    SysFreeString(connectedTo);
}

/////////////////////////////////////////////////////////////////////
// Displays the last error -- used to show the last exception  
// information.                                             
/////////////////////////////////////////////////////////////////////
void DisplayError(_com_error& error)  
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());
    static char errorStr[1024];
    _bstr_t desc = error.Description();
    if (desc.length() == 0) {
    
}
// Don't have a description string.
strcpy(errorStr, error.ErrorMessage());
int nLen = lstrlen(errorStr);

// Remove funny carriage return ctrl<M>.
if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
{
    errorStr[nLen - 2] = '\0';
}
else
{
    strcpy(errorStr, desc);
}
printf(" Error Message = %s\n", (char*) errorStr);

**GoOnline Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 207) ]

**Applies To**

- **Instrument** (see page 121) object

**Description**

Connects the user interface to a specific logic analyzer frame. The frame is locked to this user interface until it is released by calling the GoOffline (see page 207) method.

**VB Syntax**

```vb
object.GoOnline [ComputerNameOrIPAddress=""] [FrameNumber=1] [SaveChanges=False] [SaveFileName=""] [SetupOnly=False]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ComputerNameOrIPAddress</td>
<td>A <strong>String</strong> containing the frame’s computer host name as defined by the ComputerName (see page 255) property or IP address as defined by the IPAddress (see page 265) property.</td>
</tr>
<tr>
<td>FrameNumber</td>
<td>A <strong>Long</strong> that specifies the number of the frame to connect to in a multiframe configuration.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A <strong>Boolean</strong> that specifies whether changes to the current configuration should be saved.</td>
</tr>
<tr>
<td>SaveFileName</td>
<td>A <strong>String</strong> containing the name of the file to which current configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A <strong>Boolean</strong> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <strong>True</strong> – save only setup information. <strong>False</strong> – save data and setup information.</td>
</tr>
</tbody>
</table>
GoToPosition Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  • Window (see page 160) object

Description Moves the center of the window to a new position.

VB Syntax  object.GoToPosition Position

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Window (see page 160) object.</td>
</tr>
<tr>
<td>Position</td>
<td>A Variant that specifies the time of the sample to be placed at the center of the window. For time values, use the variant type Double.</td>
</tr>
</tbody>
</table>

Import Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  • Instrument (see page 121) object

Description Imports data from a file located on the instrument file system.

VB Syntax  object.Import ImportFileName [SaveChanges=False] [SaveFileName=""] [SetupOnly=False]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ImportFileName</td>
<td>A String that contains the name of the file (on the instrument file system) to be imported.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A Boolean that specifies whether changes to the current configuration should be saved.</td>
</tr>
<tr>
<td>SaveFileName</td>
<td>A String containing the name of the file to which current configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A Boolean that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: True – save only setup information. False – save data and setup information.</td>
</tr>
</tbody>
</table>

Remarks  The import file must be directly accessible by the instrument.
The only supported file format is 167xx fast binary. This file can have any suffix (*.*). To import other file types, use the ImportEx (see page 213) method.

See Also
- ImportEx (see page 213) method
- Export (see page 179) method
- ExportEx (see page 180) method

ImportEx Method

Applies To
- Instrument (see page 121) object

Description
Imports data from a file located on the instrument file system into a particular module.

VB Syntax

```vb
object.ImportEx ImportFileName DestinationName [SaveChanges=False] [SaveFileName=""] [SetupOnly=False]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ImportFileName</td>
<td>A String that contains the name of the file (on the instrument file system) to be imported.</td>
</tr>
<tr>
<td>DestinationName</td>
<td>A String that is the name of the module into which the data is imported.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A Boolean that specifies whether changes to the current configuration should be saved.</td>
</tr>
<tr>
<td>SaveFileName</td>
<td>A String containing the name of the file to which current configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A Boolean that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: True – save only setup information, False – save data and setup information.</td>
</tr>
</tbody>
</table>

Remarks
The import file must be directly accessible by the instrument.

The supported file types are:
- "Module CSV text file" (*.csv) – If DestinationName is not found, an import module is created with that name.
- "Pattern Generator CSV text file" (*.csv) – if the DestinationName is a pattern generator module.
InsertLine Method

Applies To  • PattgenModule (see page 129) object

Description  Inserts a new instruction or vector after line number.

VB Syntax  

```
object.InsertLine LineNumber, InstructionOrVector
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>A Long representing the line number after which the new instruction or vector should be inserted.</td>
</tr>
<tr>
<td>InstructionOrVector</td>
<td>A String containing the instruction or vector (see page 194) to be inserted.</td>
</tr>
</tbody>
</table>

See Also  • RemoveLine (see page 226) method

IsOnline Method

Applies To  • Instrument (see page 121) object

Description  Tells whether the user interface is connected to a logic analyzer frame.

VB Syntax  

```
object.IsOnline ComputerName
```

See Also  • Import (see page 212) method
• Export (see page 179) method
• ExportEx (see page 180) method
### Return Value

A **Boolean** indicating whether the user interface is connected to a frame. If **True** is returned, the **ComputerName** will be initialized.

### See Also
- GoOffline (see page 207) method
- GoOnline (see page 211) method

### IsTimingZoom Method

[Automation Home (see page 3)] [Objects (see page 105)] [Example]

**Applies To**
- **BusSignal** (see page 108) object

**Description**
Is this a timing zoom bus/signal?

**VB Syntax**

```vbnet
object.IsTimingZoom
```

### Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

**Return Value**
A **Boolean** indicating whether it is a timing zoom bus/signal.

### New Method

[Automation Home (see page 3)] [Objects (see page 105)] [Example]

**Applies To**
- **Instrument** (see page 121) object

**Description**
Creates a new instrument Overview.

**VB Syntax**

```vbnet
object.New [SaveChanges=False] [SaveFileName=""] [SetupOnly=False]
```

### Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A <strong>Boolean</strong> that specifies whether changes to the current configuration should be saved.</td>
</tr>
</tbody>
</table>
### Open Method

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]**

**Applies To**
- Instrument (see page 121) object

**Description**
Loads a previously saved configuration file located on the instrument file system. This will restore the instrument's settings and contents of the acquisition memory (if available). You can open either ALA or XML format configuration files.

**VB Syntax**
```
object.Open OpenFileName [SaveChanges=False] [SaveFileName=""]
[SetupOnly=False]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveFileName</td>
<td>A String that is the name of the file to which current configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A Boolean that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: True – save only setup information. False – save data and setup information.</td>
</tr>
</tbody>
</table>

**Remarks**
The configuration file must be directly accessible by the instrument.

**See Also**
- Save (see page 228) method

### PanelLock Method

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 217) ]**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>OpenFileName</td>
<td>A String that contains the name of the file located on the Instrument.</td>
</tr>
<tr>
<td>SaveChanges</td>
<td>A Boolean that specifies whether changes to the current configuration should be saved.</td>
</tr>
<tr>
<td>SaveFileName</td>
<td>A String containing the name of the file to which current configuration information should be saved.</td>
</tr>
<tr>
<td>SetupOnly</td>
<td>A Boolean that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: True – save only setup information. False – save data and setup information.</td>
</tr>
</tbody>
</table>
Applies To  • Instrument (see page 121) object

Description  Coordinates user access to the instrument front panel or remote display with other users. When locked, a full screen message is displayed indicating the instrument is currently in use by a remote COM automation program. If desired, a custom message can be shown on the local display instead of a default message. As an example, a custom message might give information as to who has the unit locked. The instrument can then be unlocked when desired.

NOTE  This method will block further execution when called from within an integrated Visual Basic for Applications (VBA) macro (because it runs from within the application).

VB Syntax  ```vbs
object.PanelLock [Message = "]"
```  

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Message</td>
<td>A String containing a custom message that will be shown on the display. If the message is empty, then a default message will be used.</td>
</tr>
</tbody>
</table>

See Also  • PanelUnlock (see page 220) method

PanelLock Example

Visual Basic

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

' Lock the instrument's front panel.
myInst.PanelLock ("Locked by Name, Phone")

' If locked, display the message.
Dim myMessage As String
If myInst.PanelLocked(myMessage) Then
    MsgBox "Remote user message: " + myMessage
End If
Unlock the instrument's front panel.
myInst.PanelUnlock

Visual C++

// This simple Visual C++ Console application demonstrates how to use
// the Agilent 168x/169x/169xx COM interface to lock and unlock the
// instrument's front panel.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// instdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA\n// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

/////////////////////////////////////////////////////////////////////
// Forward declarations.
/////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

/////////////////////////////////////////////////////////////////////
// main() entry point.
/////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[])
{
 printf("*** Main()\n");

 // Initialize the Microsoft COM/ActiveX library.
 // HRESULT hr = CoInitialize(0);

 if (SUCCEEDED(hr))
{
   try {
     // Catch any unexpected run-time errors.
     _bstr_t hostname = "mtx33"; // TODO, use your logic
     // analysis system hostname.
     printf("Connecting to instrument \"s\n", (char*) hostname);

     // Create the connect object and get the instrument object.
     AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostname);

// Lock the instrument's front panel.
pInst->PanelLock("Locked by Name, Phone");

// If locked, display the message.
BSTR myMessage;
if (pInst->GetPanelLocked(&myMessage)) {
    printf("Remote user message '%S\n", (char*) myMessage);
} SysFreeString(myMessage);

// Unlock the instrument's front panel.
pInst->PanelUnlock();
}
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();

else
{
    printf("CoInitialize failed\n");
}
return 0;
}

//////////////////////////////////////////////////////////////////////
//
// Displays the last error -- used to show the last exception
// information.
//
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");

    printf("Fatal Unexpected Error:\n");
    printf("  Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = 0;
            errorStr[nLen - 1] = 0;
        }
        printf("  Description: %s\n", errorStr);
    } else {
        printf("  Description: %s\n", desc);
    }
}

COM Automation Online Help 219
PanelUnlock Method

Applies To

- Instrument (see page 121) object

Description

Re-enables user access to the instrument front panel or remote display.

VB Syntax

object.PanelUnlock

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

See Also

- PanelLock (see page 216) method

QueryCommand Method

Applies To

- Instrument (see page 121) object
- Module (see page 128) object
- Probe (see page 135) object
- Tool (see page 152) object
- Window (see page 160) object

Description

Query for XML-based commands.

VB Syntax

object.QueryCommand Query, XMLCommand
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
<tr>
<td>Query</td>
<td>A String that contains either an XML-based query command or an XML-based filter. If a filter is specified, the query command will be set to &quot;GetAllSetup&quot;, and the output of the command will be filtered so that it only contains the attributes or tags of interest as specified by the 'Query' filter string. When a filter is used, the string returned may not be valid XML format. If the 'Query' starts with the character '&lt;', it is treated as an XML-based filter (see page 221); otherwise, it is treated as an XML-based query (see page 221) command.</td>
</tr>
</tbody>
</table>

### Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLCommand</td>
<td>A String that contains the XML-based command.</td>
</tr>
</tbody>
</table>

### Return Values

A **Boolean** indicating whether the command was successful.

**XML-Based Query**

An XML-based query command is specific to the Module, Tool, or Window. If queries are supported, the query "GetAllSetup" returns all of the setup commands.

For information about the XML-based query commands supported by a tool, see the "Tool Control, COM Automation" topic in the tool's online help.

**XML-Based Filter**

A filter can be either a fully qualified XML string or a shortened XML string. A shortened XML string does not contain end tag notation and has the following format:

```
<tag [attribute[=value]] ...> ... 
```

**Example of a fully qualified XML string filter:**

```vbnet
' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.

Dim queryOutput As String
```
myInst.QueryCommand _
"<Setup>" & _
"<Module Name=''>" & _
"<BusSignalSetup>" & _
"<BusSignal Name=''/>" & _
"</BusSignalSetup>" & _
"</Module>" & _
"</Setup>", queryOutput

queryOutput contains:

Name="MyLA-1"
Name="My Bus 1"

Example of a shortened XML string filter:

myInst.QueryCommand "<Setup><Module Name><BusSignalSetup>" + _
"<BusSignal Name>", queryOutput

queryOutput contains:

Name="MyLA-1"
Name="My Bus 1"

Both examples are equivalent and, when called, return the first module's name and its first bus/signal name.

1 If an attribute is not present at the end, the entire tag is returned. For example:

queryOutput

queryOutput contains:

<BusSignal Name='My Bus 1' Polarity='Positive' DefaultBase='Hex' Comment=''>
<Channels>Pod 1[7:0]</Channels>
</BusSignal>

2 More than one attribute can be returned. For example:

myInst.QueryCommand "<Setup><Module><BusSignalSetup>" + _
"<BusSignal Name Polarity>'", queryOutput

queryOutput contains:

Name="My Bus 1"
Polarity="Positive"

3 Tags are used to disambiguate; you can skip beginning and intermediate tag levels. Use with caution because the first tag found is used. For example:

myInst.QueryCommand "<Setup><BusSignal Name>", queryOutput

queryOutput contains:

Name="Sample Number"
NOTE

You might have expected Name to be 'My Bus 1' instead of 'Sample Number'. Because a breadth-first search is done, the BusSignal tag for 'Listing-1' is at a higher level than 'My 1690A-1'. This is why care should be taken when skipping tag levels.

4 If an '=' sign and a non-empty value are used, the tag with the given attribute value is used. This is useful when there is more than one tag with the same name at the same level, like BusSignal. For example:

```vba
myInst.QueryCommand "<Setup><Module><BusSignalSetup>" + _
"<BusSignal Name='My Bus 2' Polarity='">'', queryOutput

queryOutput contains:

Polarity="Positive"
```

5 White space is ignored. For example:

```vba
myInst.QueryCommand "< Setup > < BusSignal Name >", queryOutput
```

queryOutput contains:

Name="Sample Number"

See Also  •  "XML Format" (in the online help)

RecallTriggerByFile Method

[ Automation Home (see page 3 ) ] [ Objects (see page 105 ) ] [ Example (see page 46 ) ]

Applies To  •  AnalyzerModule (see page 106) object

Description Loads a previously saved trigger file on the instrument file system.

VB Syntax object.RecallTriggerByFile TriggerFileName

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an AnalyzerModule (see page 106) object.</td>
</tr>
<tr>
<td>TriggerFileName</td>
<td>A String that contains the name of the XML-format trigger specification file located on the instrument.</td>
</tr>
</tbody>
</table>

Remarks The RecallTriggerByFile method is a shortcut that reads an XML-format trigger specification file from the instrument's file system and sets the AnalyzerModule (see page 106) object's Trigger (see page 285) property.

See Also  •  AnalyzerModule (see page 106) object
RecallTriggerByName Method

Applies To • AnalyzerModule (see page 106) object

Description Loads a named trigger from the recall buffer.

VB Syntax object.RecallTriggerByName TriggerName

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an AnalyzerModule (see page 106) object.</td>
</tr>
<tr>
<td>TriggerName</td>
<td>A String that represents the recall buffer title name.</td>
</tr>
</tbody>
</table>

See Also • AnalyzerModule (see page 106) object

RecvFile Method

Applies To • ConnectSystem (see page 119) object

Description Copies a file from the remote logic analyzer system to your local system.

The Connect (see page 172) method must be called first to establish a connection to the logic analyzer from which the file will be copied.

VB Syntax object.RecvFile SrcFileName, DestFileName

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a ConnectSystem (see page 119) object.</td>
</tr>
<tr>
<td>SrcFileName</td>
<td>A String that is the name of the file on the remote logic analyzer system.</td>
</tr>
<tr>
<td>DestFileName</td>
<td>A String that is the name of the file on the local file system.</td>
</tr>
</tbody>
</table>

See Also • Connect (see page 172) method

Remove Method (BusSignals Object)

Applies To • BusSignals (see page 113) object

Description Removes a bus/signal from the BusSignals collection.
**VB Syntax**  
```
BusSignals (see page 113).Remove Name
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A String containing the name of the bus/signal to be removed.</td>
</tr>
</tbody>
</table>

**Remove Method (Markers Object)**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

**Applies To**  
- Markers (see page 123) object

**Description**  
Removes a marker from the Markers collection.

**VB Syntax**  
```
object.Remove Name
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Markers (see page 123) object.</td>
</tr>
<tr>
<td>Name</td>
<td>A String containing the name of the marker to be removed.</td>
</tr>
</tbody>
</table>

**RemoveAll Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

**Applies To**  
- Markers (see page 123) object

**Description**  
Removes all markers from the Markers collection.

**VB Syntax**  
```
object.RemoveAll
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Markers (see page 123) object.</td>
</tr>
</tbody>
</table>

**RemoveXML Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

**Applies To**  
- Markers (see page 123) object

**Description**  
Removes multiple Marker (see page 123) objects from the Markers (see page 123) collection using an XML string.

**VB Syntax**  
```
object.RemoveXML XMLMarkers
```
RemoveLine Method

Applies To
- PattgenModule (see page 129) object

Description
Removes the instruction or vector at line number.

VB Syntax
object.RemoveLine LineNumber

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>XMLMarkers</td>
<td>An &quot;XML format&quot; (in the online help)String containing the markers to remove (see &quot;&lt;Markers&gt; element&quot; (in the online help)).</td>
</tr>
</tbody>
</table>

Reset Method (PattgenModule Object)

Applies To
- PattgenModule (see page 129) object

Description
Resets the current line number to the first line.

VB Syntax
object.Reset

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
</tbody>
</table>

See Also
- InsertLine (see page 214) method
- Run (see page 228) method
- Stop (see page 236) method
Resume Method (PattgenModule Object)

Applies To • PattgenModule (see page 129) object

Description Resumes running the pattern generator from the current line number.

VB Syntax object.Resume

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
</tbody>
</table>

See Also • Run (see page 228) method  
• Stop (see page 236) method  
• Step (see page 235) method  
• Reset (see page 226) method

Run Method (Instrument Object)

Applies To • Instrument (see page 121) object

Description Starts running all modules.

VB Syntax object.Run [Repetitive=False]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Repetitive</td>
<td>A Boolean indicating the number of times the module(s) are started. True – run continuously until the Stop (see page 236) method is called. False – run only once.</td>
</tr>
</tbody>
</table>

See Also • Stop (see page 236) method  
• Status (see page 280) property  
• WaitComplete (see page 238) method
Run Method (PattgenModule Object)

Applies To • PattgenModule (see page 129) object

Description Starts running the pattern generator.

VB Syntax object.Run [Repetitive=False]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>Repetitive</td>
<td>A Boolean indicating the number of times the module is started. True – run continuously until the Stop (see page 236) method is called. False – run only once.</td>
</tr>
</tbody>
</table>

See Also • Stop (see page 236) method
• Step (see page 235) method
• Resume (see page 227) method
• Reset (see page 226) method
• Status (see page 280) property
• WaitComplete (see page 238) method

Save Method

Applies To • Instrument (see page 121) object

Description Saves the current configuration to a file on the instrument file system.

VB Syntax object.Save SaveFileName [SetupOnly=False]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>
SendFile Method

- Apply: ConnectSystem (see page 119) object
- Description: Copies a file from your local system to the remote logic analyzer system. The Connect (see page 172) method must be called first to establish a connection to the logic analyzer to which the file will be copied.
- VB Syntax: `object.SendFile SrcFileName, DestFileName`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a ConnectSystem (see page 119) object.</td>
</tr>
<tr>
<td>SrcFileName</td>
<td>A String that is the name of the file on the local file system.</td>
</tr>
<tr>
<td>DestFileName</td>
<td>A String that is the name of the file on the remote logic analyzer system.</td>
</tr>
</tbody>
</table>

SetGroupCaption Method

- Apply: VbaViewChartData (see page 157) object
- Description: Sets the caption associated with a group (row).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a ConnectSystem (see page 119) object.</td>
</tr>
<tr>
<td>SrcFileName</td>
<td>A String that is the name of the file on the local file system.</td>
</tr>
<tr>
<td>DestFileName</td>
<td>A String that is the name of the file on the remote logic analyzer system.</td>
</tr>
</tbody>
</table>
### SetGroupCaption Method

**VB Syntax**

```
object.SetGroupCaption Group, Caption
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
<tr>
<td>Group</td>
<td>A <code>Long</code> representing the group (row) on which to set the caption.</td>
</tr>
<tr>
<td>Caption</td>
<td>A <code>String</code> that is the caption to be set.</td>
</tr>
</tbody>
</table>

**See Also**
- GetGroupCaption (see page 193) method

### SetLine Method

**VB Syntax**

```
object.SetLine LineNumber, InstructionOrVector
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>A <code>Long</code> representing the line number at which the instruction or vector</td>
</tr>
<tr>
<td>InstructionOrVector</td>
<td>A <code>String</code> containing the instruction or vector (see page 194) to be inserted.</td>
</tr>
</tbody>
</table>

**See Also**
- GetLine (see page 194) method

### SetLineLabel Method

**VB Syntax**

```
object.SetLineLabel LineNumber, LabelName, Value
```

**Applies To**
- PattgenModule (see page 129) object

**Description**
Sets a vector's label value at line number.

**See Also**
- GetLine (see page 194) method
See Also  •  GetLineLabel (see page 196) method

SetValue Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To  •  VbaViewChartData (see page 157) object

Description  Sets an individual value in the chart array.

VB Syntax  object.SetValue (Group, Index, Value)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>A Long representing the line number from which to get the instruction or vector.</td>
</tr>
<tr>
<td>LabelName</td>
<td>A String name of label whose value you wish to set.</td>
</tr>
<tr>
<td>Value</td>
<td>A String representing the label's value.</td>
</tr>
</tbody>
</table>

See Also  •  SetValueArray (see page 231) method

SetValueArray Method

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To  •  VbaViewChartData (see page 157) object

Description  Sets an array of values in the chart array starting at index 0.

VB Syntax  object.SetValueArray (Group, ValueArray)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
<tr>
<td>Group</td>
<td>A Long representing the group (row) on which to set the value.</td>
</tr>
<tr>
<td>Index</td>
<td>A Long representing the index (column) on which to set the value.</td>
</tr>
<tr>
<td>Value</td>
<td>A Variant that is the value to set in the chart.</td>
</tr>
</tbody>
</table>
See Also  
- `SetValue` (see page 231) method

**SetValueCaption Method**

```vbnet
object.SetValueCaption Index, Caption
```

### Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
<tr>
<td>Group</td>
<td>A <code>Long</code> representing the group (row) on which to set the value array.</td>
</tr>
<tr>
<td>ValueArray</td>
<td>An array of <code>Variants</code> that are the values to set in the chart.</td>
</tr>
</tbody>
</table>

**See Also**  
- `GetValueCaption` (see page 206) method

**SimpleTrigger Method**

```vbnet
object.SimpleTrigger [OnEvent = "Anything"], [Occurs=1], [StoreEvent="Anything"]
```

### Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartData (see page 157) object.</td>
</tr>
<tr>
<td>Index</td>
<td>A <code>Long</code> representing the index (column) on which to set the caption.</td>
</tr>
<tr>
<td>Caption</td>
<td>A <code>String</code> that is the caption to be set.</td>
</tr>
</tbody>
</table>

**See Also**  
- `AnalyzerModule` (see page 106) object

**Description**  
Trigger on a simple condition with optional occurrence and storage qualification. Default triggers on anything and stores everything.
If the analyzer is in timing mode, the only valid value for `StoreEvent` is "Anything".

**Remarks**

**See Also**
- `RecallTriggerByFile` (see page 223) method
- `RecallTriggerByName` (see page 224) method

**Simple Trigger/Find Event**

An Event contains either a `ComboValue`, the String "Anything", or the String "Nothing". For the Find (see page 184) method, an Event can only contain a `ComboValue`.

A `ComboValue` contains a `BusSignal` optionally followed by the String "And" or "Or" followed by another `BusSignal`.

A `BusSignal` contains a bus/signal name followed by a `Relational` followed by a `Range`, `Value`, or `Edge`.

A `Relational` string can be:
- "=". "!=". ">". "><". ">=". or "><="

Additional `Relational` strings used with the Find (see page 184) method are:
- "Entering" – the first sample of one or more consecutive samples that match the pattern. (By comparison, the "=" equals operator considers every sample that matches the pattern as an occurrence.)
- "Exiting" – the sample after one or more consecutive samples that match the pattern.
- "Transitioning" – entering or exiting one or more consecutive samples that match the pattern.
A **Range** contains a **Value** followed by the **String** ".." followed by another **Value**. Range values cannot contain don't care digits. A range can only be used with the **Relational** strings "==" and "!=".

A **Value** string contains a number with optional don't care digits or an edge. A number base is required; use the following prefixes:

- h — hexadecimal
- o — octal
- b — binary
- d — decimal

An **Edge** string begins with "e" followed by any combination of the following upper-case characters:

- X — don't care
- R — rising edge
- F — falling edge
- E — either edge
- G — glitch (only valid for the SimpleTrigger (see page 232) method)

**Remarks**  **Value** strings are case insensitive.

When an **Edge** string is used, the **BusSignal's Relational** string must be "=". When used with the Find (see page 184) method, an **Edge** specification can only be one bit wide, and you cannot specify a glitch. When used with the SimpleTrigger (see page 232) method, an **Edge** is only valid when the analyzer is in timing mode.

Bus/signal names that contain " And " (with spaces before and after), " Or " (with spaces before and after), or end in a non-alphanumeric character will confuse the parser and produce unexpected results.

**Example**

**Value examples:**

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hFFXX0022</td>
<td>Hex number with 2 don't care digits (8 don't care bits).</td>
</tr>
<tr>
<td>o7777xxxx</td>
<td>Octal number with 4 don't care digits (12 don't care bits).</td>
</tr>
<tr>
<td>b10110110xxxx0000</td>
<td>Binary number with 4 don't care bits.</td>
</tr>
</tbody>
</table>

**Range examples:**

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hff00..hffff</td>
<td>Range from hex f00 to ffff.</td>
</tr>
</tbody>
</table>
**Step Method**

Applies To
- PattgenModule (see page 129) object

Description
Steps the pattern generator from the current line number.

VB Syntax
object.\texttt{Step} [\texttt{Count}=1]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
<tr>
<td>Count</td>
<td>A Long indicating the number of vectors the pattern generator should output.</td>
</tr>
</tbody>
</table>

See Also
- Run (see page 228) method
- Stop (see page 236) method

**Edge examples:**

<table>
<thead>
<tr>
<th>Edge specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXXXXRFEG</td>
<td>Edge specification with 4 don't care bits, then rising, falling, either, and glitch bits.</td>
</tr>
<tr>
<td>eR</td>
<td>A rising edge on a signal (one bit).</td>
</tr>
</tbody>
</table>

**Event examples:**

<table>
<thead>
<tr>
<th>Event specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR=ffffxxxx</td>
<td>Specifies a simple bus/signal value.</td>
</tr>
<tr>
<td>ADDR=ffff0000..ffffffffff</td>
<td>Specifies the same simple bus/signal value as a range.</td>
</tr>
<tr>
<td>ADDR=ffff0000..ffffffffff And DATA=eXXXXRXXXX</td>
<td>Specifies a rising edge in bit 5 of DATA while ADDR is within the hex range ffff0000 to ffffff.</td>
</tr>
<tr>
<td>ADDR=ffffxx And DATA=h055</td>
<td>Specifies AND'ing two bus/signal values.</td>
</tr>
</tbody>
</table>

**Trigger Event example:**

<table>
<thead>
<tr>
<th>Event specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anything</td>
<td>Specifies any value.</td>
</tr>
</tbody>
</table>

**Find Event example:**

<table>
<thead>
<tr>
<th>Event specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDR entering hff00</td>
<td>Specifies the first sample of one or more consecutive samples whose ADDR is hFF00.</td>
</tr>
</tbody>
</table>

See Also
- SimpleTrigger (see page 232) method
- Find (see page 184) method
Stop Method (Instrument Object)

Applies To
- Instrument (see page 121) object

Description
Stops all currently running data acquisition modules (that is, not pattern generator modules). If you want to stop a pattern generator module, call the PattgenModule (see page 129) object's Stop (see page 236) method.

NOTE
If self tests are running, this will also stop and close the Self Test dialog.

VB Syntax
object.Stop

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

See Also
- Run (see page 227) method  
- Status (see page 280) property

Stop Method (PattgenModule Object)

Applies To
- PattgenModule (see page 129) object

Description
Stops the pattern generator if it is currently running.

VB Syntax
object.Stop

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
</tbody>
</table>

See Also
- Step (see page 235) method  
- Resume (see page 227) method  
- Run (see page 228) method  
- Reset (see page 226) method
**TestAll Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 150) ]

**Applies To**
- **SelfTest** (see page 149) object

**Description**
Runs an instrument's self-tests.

**VB Syntax**
`object.TestAll`

**Return Values**
The TestAll method returns one of the following run result values:
- "RUN_RESULT_INIT_FAILED"
- "RUN_RESULT_FAILED"
- "RUN_RESULT_INCOMPLETE"
- "RUN_RESULT_PASSED"

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a SelfTest (see page 149) object.</td>
</tr>
</tbody>
</table>

**VBADisplayHelpTopic Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- **Instrument** (see page 121) object

**Description**
Displays the help page and topic for an installed VBA project.

**VB Syntax**
`object.VBADisplayHelpTopic ProjectName [HelpTopic=""]`

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>ProjectName</td>
<td>A String that is the name of the VBA project. This is the same VBA project name that appears in the Project.xml file within the .zip file that contains the project sources.</td>
</tr>
<tr>
<td>HelpTopic</td>
<td>A String that is the name of a topic in the help file.</td>
</tr>
</tbody>
</table>

**VBARunMacro Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- **Instrument** (see page 121) object
**Description**  Runs the specified VBA macro as if that macro was selected in the Macros dialog box.

**VB Syntax**  
```vb
object.VBARunMacro MacroName
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>MacroName</td>
<td>A String that is the name of the VBA macro to run.</td>
</tr>
</tbody>
</table>

**VBARunRPICommand Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**  
- Instrument (see page 121) object

**Description**  Runs an ASCII RPI command in VBA.

**VB Syntax**  
```vb
object.VBARunRPICommand Command
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Command</td>
<td>A String containing the ASCII RPI command to run.</td>
</tr>
</tbody>
</table>

**Return Value**  
A Variant that is the return value of the ASCII RPI command.

**WaitComplete Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 239) ]

**Applies To**  
- AnalyzerModule (see page 106) object
- Instrument (see page 121) object
- Module (see page 128) object
- PattgenModule (see page 129) object

**Description**  Waits until a measurement completes or a timeout (in seconds) occurs. Executing the Instrument (see page 121) object's WaitComplete method waits for all data acquisition modules (that is, not pattern generator modules), tools, and viewers to complete their measurements. Executing a Module (see page 128) object's WaitComplete method waits for the module to complete its measurement.
**VB Syntax**

```vbnet
object.WaitComplete [Seconds=-1]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
<tr>
<td>Seconds</td>
<td>A <code>Long</code> containing the maximum number of seconds to wait for the measurement to complete. <strong>Note:</strong> If set to -1, this method does not return until the specified measurement completes; therefore, we strongly recommend you make this value &gt;= 0.</td>
</tr>
</tbody>
</table>

**WaitComplete Example**

**Visual Basic**

```vbnet
Private Sub Command1_Click()

' If WaitComplete times out, a run-time error occurs.
On Error GoTo ErrorHandler

' When using Visual Basic outside of the Agilent Logic Analyzer application, you must create the Connect object (see page 118) and use it to access the Instrument object. In this example, "myInst" represents the Instrument object.
'
' When "using the Advanced Customization Environment (ACE)" (in the online help), the Instrument object is already created and is globally accessible using "AgtLA". In this example, substitute "myInst" with "AgtLA" to access the global Instrument object in VBA.
'
' Load the configuration file.
myInst.Open ("c:\LA\Configs\Test1.ala")

' Load the logic analyzer trigger file.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.GetModuleByName("My 16756A-1")
myAnalyzer.RecallTriggerByFile ("c:\LA\Triggers\Test1_TrigSpec.xml")

' Run the measurement, wait for it to complete.
myInst.Run
myInst.WaitComplete (20)

' Notify when measurement is complete.
MsgBox "Measurement complete."

Exit Sub

ErrorHandler:

' Handle the error that occurs if WaitComplete times out.
Select Case Err.Number
Case -2147352567
  myInst.Stop
  MsgBox "WaitComplete timed out, measurement stopped."
```
Visual C++

// This simple Visual C++ Console application demonstrates how to use the Agilent 168x/169x/169xx COM interface to wait until a measurement is complete.

// This project was created in Visual C++ Developer. To create a similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA \ COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect to (search for "TODO" below).
//
#include "stdafx.h"

/************************************************************************* /
// Forward declarations.
/************************************************************************* /
void DisplayError(_com_error& err);

/************************************************************************* /
// main() entry point.
/************************************************************************* /
int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    // Initialize the Microsoft COM/ActiveX library.
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr))
    {
        try { // Catch any unexpected run-time errors.
            _bstr_t hostname = "mtx33"; // TODO, use your logic
            // analysis system hostname.
        }
printf("Connecting to instrument \'%s\'\n", (char*) hostname);

// Create the connect object and get the instrument object.
AgtLA::IConnectPtr pConnect =
    AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
AgtLA::IInstrumentPtr pInst =
    pConnect->GetInstrument(hostname);

// Load the configuration file.
_bstr_t configFile = "C:\LA\Configs\config.ala";
printf("Loading the config file \'%s\'\n", (char*) configFile);
pInst->Open(configFile, FALSE, "", TRUE);

// Set up the trigger.
_bstr_t moduleName = "MPC860 Demo Board";
AgtLA::IAnalyzerModulePtr pAnalyzer =
    pInst->GetModuleByName(moduleName);
pAnalyzer->SimpleTrigger("ADDR=0", 1, "Anything");

// Run the measurement, wait for it to complete.
pInst->Run(FALSE);
try {
    pInst->WaitComplete(20);
    printf("Measurement complete.\n");
} catch (_com_error& e) {
    switch (e.Error()) {
    case 0x80020009:
        pInst->Stop();
        printf("Inner WaitComplete timed out, ");
        printf("measurement stopped.\n");
        break;
    default:
        throw;
        break;
    }
    printf("End of program.\n");
} catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
else
{
    printf("CoInitialize failed\n");
}

return 0;

/////////////////////////////////////////////////////////////////////
//
// Displays the last error -- used to show the last exception
// information.
//
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }

    printf(" Error Message = %s\n", (char*) errorStr);
}

**WriteOutput Method**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**Applies To**

- VbaViewWindow (see page 160) object

**Description**

Writes a string to the output window.

**VB Syntax**

```vb
object.WriteOutput String
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWindow (see page 160) object.</td>
</tr>
<tr>
<td>String</td>
<td>A <em>String</em> to be written to the VbaView window.</td>
</tr>
</tbody>
</table>
Properties

- Activity Property (see page 245)
- Axis Property (see page 246)
- AxisBase Property (see page 246)
- BackgroundColor Property (see page 247)
- BitSize Property (see page 248)
- BitSize Property (of VbaViewChartAxis object) (see page 248)
- Bold Property (see page 248)
- BusSignalData Property (see page 249)
- BusSignalType Property (see page 249)
- BusSignalDifferences Property (see page 250)
- BusSignals Property (see page 250)
- ByteSize Property (see page 251)
- Caption Property (see page 251)
- CardModels Property (see page 252)
- Channels Property (see page 252)
- Chart Property (see page 253)
- ChartType Property (see page 253)
- Color Property (see page 254)
- Comments Property (see page 255)
- ComputerName Property (see page 255)
- Count Property (see page 256)
- CreatorName Property (see page 259)
- Data Property (see page 260)
- DataType Property (see page 260)
- Description Property (see page 260)
- Differences Property (see page 261)
- EndSample Property (see page 261)
- EndTime Property (see page 262)
- FaceName Property (see page 262)
- Font Property (see page 262)
- Found Property (see page 263)
- Frame Property (see page 263)
- Frames Property (see page 263)
- HasLegend Property (see page 264)
- HasTitle Property (see page 264)
- Instrument Property (see page 265)
- IPAddress Property (see page 265)
- Item Property (see page 266)
- Legend Property (see page 267)
- Markers Property (see page 267)
- Model Property (see page 268)
- Modules Property (see page 268)
- Name Property (see page 269)
- NumLines Property (see page 270)
- OccurrencesFound Property (see page 270)
- Options Property (see page 271)
- Overview Property (see page 271)
- PanelLocked Property (see page 272)
- Polarity Property (see page 272)
- Position Property (see page 272)
- Position Property (of VbaViewChartLegend object) (see page 273)
- Probes Property (see page 274)
- Reference Property (see page 274)
- RemoteComputerName Property (see page 274)
- RemoteUserName Property (see page 275)
- RunningStatus Property (see page 275)
- SampleDifferences Property (see page 276)
- SampleNum Property (see page 277)
- SelfTest Property (see page 277)
- Setup Property (see page 277)
- Size Property (see page 278)
- Slot Property (see page 278)
- StartSample Property (see page 279)
- StartTime Property (see page 279)
- Status Property (see page 280)
- StatusMsg Property (see page 281)
- SubrowFound Property (see page 281)
- Symbols Property (see page 281)
- TargetControlPort Property (see page 282)
- TextColor Property (see page 283)
- TimeFound Property (see page 283)
- TimeFoundString Property (see page 284)
- Title Property (see page 284)
- Tools Property (see page 284)
- Trigger Property (see page 285)
- Type Property (see page 285)
- Value Property (see page 286)
- VBAVersion Property (see page 287)
- VBE Property (see page 287)
- Version Property (see page 287)
- WebBrowser Property (see page 288) (for the VbaViewWindow object)
- WebBrowser Property (see page 288) (for the VbaViewWebBrowser object)
- Windows Property (see page 289)
- _NewEnum Property (see page 289)

**Activity Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- BusSignal (see page 108) object

**Description**
Gets the activity indicators of the bus/signal.

**VB Syntax**
object.Activity

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

**Remarks**
The **Activity** property has the **String** type.

There is an activity indicator character for each signal, which can be:
- - (dash) – no activity
- H – activity level high
- L – activity level low
- T – activity level transition
- B – activity level is both high and low
### Axis Property

**Applies To**  
- VbaViewChart (see page 156) object

**Description**  
Gets the chart axis given an axis type.

**VB Syntax**  
object.Axis AxisType

**Parameters**  
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
<tr>
<td>AxisType</td>
<td>An AgtChartAxisType enumerated type value that identifies the X or Y axis. See the description below.</td>
</tr>
</tbody>
</table>

**Remarks**  
The Axis property has the VbaViewChartAxis (see page 157) object type. The AxisType parameter can have the following values:

<table>
<thead>
<tr>
<th>AgtChartAxisType</th>
<th>Enum Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtChartAxisTypeX</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AgtChartAxisTypeY</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### AxisBase Property

**Applies To**  
- VbaViewChartAxis (see page 157) object

**Description**  
Gets or sets the chart axis base.

Formatting axis values in different bases is only supported when the chart type is AgtChartTypeXYScatter (see the VbaViewChart (see page 156) object's ChartType (see page 253) property).

**VB Syntax**  
object.AxisBase [=Base]

**Parameters**  
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartAxis (see page 157) object.</td>
</tr>
<tr>
<td>Base</td>
<td>An AgtAxisBase enumerated type value that can be one of the values described below.</td>
</tr>
</tbody>
</table>

**Remarks**  
The AxisBase property can have the following values:
BackgroundColor Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To • Marker (see page 123) object

Description Gets or sets the marker background color.

' Display the marker background color.
Dim myBackgroundColor As Long
myBackgroundColor = myMarker.BackgroundColor
MsgBox Str(myBackgroundColor)

' Set the marker background color to green.
myBackgroundColor = &H0000FF00
myMarker.BackgroundColor = myBackgroundColor

VB Syntax object.BackgroundColor [=Color]

Parameters Definition

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Marker (see page 123) object.</td>
</tr>
<tr>
<td>Color</td>
<td>A Long value that is the marker background color.</td>
</tr>
</tbody>
</table>

Remarks The BackgroundColor property has the Long type.

Color values have the following hexadecimal form: 0x00BBGGRR. The low-order byte (RR) contains a value for the relative intensity of red; the second byte (GG) contains a value for green; and the third byte (BB) contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is &HFF. The color white is &H00FFFFFF, black is &H00000000, and red is &H000000FF.
**BitSize Property**

- **Applies To**: BusSignal (see page 108) object
- **Description**: Gets the number of channels in the bus/signal.
- **VB Syntax**: `object.BitSize`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

- **Remarks**: The BitSize property has the Long type.
- **See Also**: ByteSize (see page 251) property

**BitSize Property (of VbaViewChartAxis)**

- **Applies To**: VbaViewChartAxis (see page 157) object
- **Description**: Gets or sets the width of the data in bits. This is used to format the Axis values.
- **VB Syntax**: `object.BitSize [=BitSize]`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an VbaViewChartAxis (see page 157) object.</td>
</tr>
<tr>
<td>BitSize</td>
<td>A Long value that specifies the width of the data in bits.</td>
</tr>
</tbody>
</table>

- **Remarks**: The BitSize property has the Long type.
- **See Also**: AxisBase (see page 246) property

**Bold Property**

- **Applies To**: VbaViewChartFont (see page 158) object
- **Description**: Gets or sets the text thickness.
- **VB Syntax**: `object.Bold [=Bold]`
**Bold** property has the **Boolean** type.

### BusSignalData Property

- **Applies To** BusSignal (see page 108) object
- **Description** Gets the data associated with a bus/signal.
- **VB Syntax** `object.BusSignalData`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

**Remarks** The **BusSignalData** property has the **BusSignalData** (see page 108) object type.

### BusSignalType Property

- **Applies To** BusSignal (see page 108) object
- **Description** Gets the type of bus/signal.
- **VB Syntax** `object.BusSignalType`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

**Remarks** The **BusSignalType** property can have the following values:

<table>
<thead>
<tr>
<th>AgtBusSignalType</th>
<th>Enum Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtBusSignalProbed</td>
<td>1</td>
<td>The bus/signal is associated with a physically probed connection.</td>
</tr>
</tbody>
</table>
The Activity (see page 245), Channels (see page 252), and Polarity (see page 272) properties are only valid when the BusSignalType property is AgtBusSignalProbed.

### BusSignalDifferences Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]

Appplies To
- SampleDifference (see page 149) object

Description
Gets a collection of all the buses/signals with differences for this sample.

VB Syntax
```vbnet
object.BusSignalDifferences
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a SampleDifference (see page 149) object.</td>
</tr>
</tbody>
</table>

Remarks
The BusSignalDifferences property has the BusSignalDifferences (see page 109) collection object type. Each item in the collection is a BusSignalDifference (see page 109) object.

### BusSignals Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 139) ]

Appplies To
- Module (see page 128) object
- Tool (see page 152) object
- Window (see page 160) object
Description
Gets a collection of the module's defined buses/signals.

VB Syntax
object.BusSignals

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the Applies to line.</td>
</tr>
</tbody>
</table>

Remarks
The **BusSignals** property has the **BusSignals (see page 113)** collection object type. Each item in the collection is a BusSignal (see page 108) object.

**ByteSize Property**

- [ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 139) ]

Applies To
- **BusSignal (see page 108)** object

Description
Gets the size of the bus/signal in bytes.

VB Syntax
object.ByteSize

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

Remarks
The **ByteSize** property has the **Long** type.

If the size of the bus is not a multiple of 8, it will be rounded to the next byte. For example, if a bus is 17 bits wide, 3 bytes will be returned. This property is useful when attempting to extract data from a byte array (raw). See the GetDataBySample (see page 189) method.

See Also
- **BitSize (see page 248)** property

**Caption Property**

- [ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To
- **VbaViewChartTitle (see page 159)** object

Description
Gets or sets the chart title caption.

VB Syntax
object.Caption [=Caption]
### CardModels Property

Applies To: Module (see page 128) object

Description:

Gets the card model numbers.

VB Syntax:

```vbnet
object.CardModels
```

#### Remarks

The **CardModels** property has the **String** type.

### Channels Property

Applies To: BusSignal (see page 108) object

Description:

Gets the channels defined in the bus/signal.

VB Syntax:

```vbnet
object.Channels
```
The Channels property has the String type.

### Chart Property

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWindow (see page 160) object.</td>
</tr>
</tbody>
</table>

**Channels**

A String containing MultiplePodChannels or the String "None" if no channels are assigned.

MultiplePodChannels contains a comma separated list of PodChannels.

PodChannels contains a PodNumber followed by the String "]" followed by a comma separated list of individual channel Number(s) and NumberRange(s) followed by the String "]".

*Note:* Pod channels normally are in MSB to LSB notation unless you are trying to reorder channels.

NumberRange contains a Number followed by the String ":" followed by a Number.

PodChannels Example:

Pod 1[9:7, 5, 3:1] — this bus consists of 1 single channel Number and 2 NumberRange’s for a total of 7 channels. They are Pod 1[9], Pod 1[8], Pod 1[7], Pod 1[5], Pod 1[3], Pod 1[2], Pod 1[1].

MultiplePodChannels Example:

Pod 2[3, 1], Pod 3[10, 5:3] — this bus consists of 3 single channel Numbers and 1 NumberRange for a total of 6 channels Pod 2[3], Pod 2[1], Pod 3[10], Pod 3[5], Pod 3[4], Pod 3[3].

**Remarks**

The Chart property has the VbaViewChart (see page 156) object type.

### ChartType Property

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWindow (see page 160) object.</td>
</tr>
</tbody>
</table>

**Remarks**

The Chart property has the VbaViewChart (see page 156) object type.
Applies To  •  VbaViewChart (see page 156) object

Description  Gets or sets the chart type.

VB Syntax  object.ChartType [=Type]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
<tr>
<td>Type</td>
<td>An AgtChartType enumerated type value that can be one of the values described below.</td>
</tr>
</tbody>
</table>

Remarks  The ChartType property can have the following values:

<table>
<thead>
<tr>
<th>AgtChartType</th>
<th>Enum Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgtChartTypeNone</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeLine</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeLineOnly</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeXYScatter</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeHorizontalBar</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeVerticalBar</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypePie</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeStackedVerticalBar</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>AgtChartTypeStackedHorizontalBar</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Color Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To  •  VbaViewChartFont (see page 158) object

Description  Gets or sets the text color.

VB Syntax  object.Color [=Color]
The Color property has the Long type.

Color values have the following hexadecimal form: 0x00BBGGRR. The low-order byte (RR) contains a value for the relative intensity of red; the second byte (GG) contains a value for green; and the third byte (BB) contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is &HFF. The color white is &H00FFFFFF, black is &H00000000, and red is &H000000FF.

Comments Property

The Comments property has the String type.

ComputerName Property

The ComputerName property has the String type.
### Count Property

- **Parameters**: `Object`  
  - An expression that evaluates to a `Frame` object. The `ComputerName` property is the default property of the `Frame` object. Accordingly, you do not have to reference `ComputerName` explicitly, as shown in the syntax.

- ** Applies To**: 
  - `BusSignalDifferences` object
  - `BusSignals` object
  - `Frames` object
  - `Markers` object
  - `Modules` object
  - `Probes` object
  - `SampleDifferences` object
  - `Tools` object
  - `Windows` object

- **Description**: Gets the number of items in a collection.

- **VB Syntax**: `object.Count` 

- **Remarks**: The `Count` property has the `Long` type.

- **See Also**:  
  - `Item` property

---

**Item and Count Example**

The following example displays the channels for each bus/signal in the `BusSignals` collection:

**Visual Basic**

```vbnet
Dim myBusSignalChannels As String
Dim myBusSignal As AgtLA.BusSignal
For i = 0 To myInst.GetModuleByName("My 1690A-1").BusSignals.Count - 1
    Set myBusSignal = myInst.GetModuleByName("My 1690A-1").BusSignals(i)
    myBusSignalChannels = myBusSignalChannels & ", " & myBusSignal.Name
Next i
```
' Add the bus/signal name and channels to the string.
myBusSignalChannels = myBusSignalChannels + vbCrLf
myBusSignalChannels = myBusSignalChannels + "Bus/signal: " + _
    myBusSignal.Name
myBusSignalChannels = myBusSignalChannels + ", Channels: " + _
    myBusSignal.Channels
Next
MsgBox "Bus/signal names and channels: "+ vbCrLf + _
    myBusSignalChannels

Visual C++

// This simple Visual C++ Console application demonstrates how to
// use the Agilent 168x/169x/169xx COM interface to display the
// channels for all buses/signals.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA\
// COM Automation/agClientSvr.dll"
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

////////////////////////////////////////////////////////////////////
// Forward declarations.
////////////////////////////////////////////////////////////////////

void DisplayError(_com_error& err);

////////////////////////////////////////////////////////////////////
// main() entry point.
////////////////////////////////////////////////////////////////////

int main(int argc, char* argv[])
{
    printf("*** Main()\n");

    //
    // Initialize the Microsoft COM/ActiveX library.
    //
    HRESULT hr = CoInitialize(0);

    if (SUCCEEDED(hr))
    {
try { // Catch any unexpected run-time errors.
    _bstr_t hostname = "mtx33"; // TODO, use your logic
    // analysis system hostname.
    printf("Connecting to instrument %s\n", (char*) hostname);

    // Create the connect object and get the instrument object.
    AgtLA::IConnectPtr pConnect =
        AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
    AgtLA::IInstrumentPtr pInst =
        pConnect->GetInstrument(hostname);

    // Load the configuration file.
    _bstr_t configFile = "C:\LA\Configs\config.ala";
    printf("Loading the config file %s\n", (char*) configFile);
    pInst->Open(configFile, FALSE, "", TRUE);

    // Display the channels for all probed bus/signal.
    _bstr_t moduleName = "MPC860 Demo Board";
    _bstr_t busSignal;
    _bstr_t channels;
    AgtLA::IAnalyzerModulePtr pAnalyzer =
        pInst->GetModuleByName(moduleName);
    AgtLA::IBusSignalsPtr pBusSignals = pAnalyzer->GetBusSignals();

    for (long i = 0; i < pBusSignals->GetCount(); i++)
    {
        AgtLA::IBusSignalPtr pBusSignal = pBusSignals->GetItem(i);
        AgtLA::AgtBusSignalType busSignalType =
            pBusSignal->GetBusSignalType();

        if (busSignalType == AgtLA::AgtBusSignalProbed) {
            busSignal = pBusSignal->GetName();
            channels = pBusSignal->GetChannels();
            printf("Bus/signal %s, channels %s.\n", 
                   (char*) busSignal, (char*) channels);
        }
    }
    catch (_com_error& e) {
        DisplayError(e);
    }

    // Uninitialize the Microsoft COM/ActiveX library.
    CoUninitialize();
} else
{
    printf("CoInitialize failed\n");
}

return 0;

// ////////////////////////////////////////////////////////////////////////////

// Displays the last error -- used to show the last exception information.
void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");

    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0)
    {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);

        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd))
        {
            errorStr[nLen - 2] = '\0';
        }
    }
    else
    {
        strcpy(errorStr, desc);
    }

    printf(" Error Message = %s\n", (char*) errorStr);
}

CreatorName Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To
- BusSignal (see page 108) object

Description
Gets the name of the module, tool, or viewer that created this bus/signal.

VB Syntax
object.CreatorName

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

Remarks
The CreatorName property has the String type.
Data Property

Applies To • VbaViewChart (see page 156) object
Description Gets the chart axis given an axis type.
VB Syntax object.Data

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
</tbody>
</table>

Remarks The Data property has the VbaViewChartData (see page 157) object type.

DataType Property

Applies To • SampleBusSignalData (see page 138) object
Description Gets the recommended bus/signal data type.
Note that, while this is the recommended data type, other data types can be used for uploading the data.
VB Syntax object.DataType

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a SampleBusSignalData (see page 138) object.</td>
</tr>
</tbody>
</table>

Remarks The DataType property can have values defined by the AgtDataType enumerated type. See DataTypes and Return Values (see page 190).

See Also • BusSignalType (see page 249) property

Description Property

Applies To • Frame (see page 120) object
• Module (see page 128) object
Description Gets a description of the logic analyzer frame or module.
VB Syntax  object.Description

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Frame (see page 120) or Module (see page 128) object.</td>
</tr>
</tbody>
</table>

Remarks  The Description property has the String type.

There is no defined format for the description string; therefore, do not parse this string to extract specific information.

Differences Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  • CompareWindow (see page 118) object

Description  Gets the number of differences found on the last comparison.

VB Syntax  object.Differences

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a CompareWindow (see page 118) object.</td>
</tr>
</tbody>
</table>

Remarks  The Differences property has the Long type.

A comparison can be done directly by calling the Execute (see page 178) method or indirectly when its input data changes and comparisons are enabled in the user interface.

See Also  • Execute (see page 178) method

EndSample Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  • SampleBusSignalData (see page 138) object

Description  Gets the data's ending sample number relative to trigger.

VB Syntax  object.EndSample

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData (see page 138) object.</td>
</tr>
</tbody>
</table>
Remarks  The **EndTime** property has the **Double** type.

See Also  • StartTime (see page 279) property

**EndTime Property**

- Applies To  • **SampleBusSignalData** (see page 138) object
- Description  Gets the data's ending time (in seconds) relative to trigger.
- VB Syntax  `object.EndTime`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>An expression that evaluates to an <strong>SampleBusSignalData</strong> object.</td>
</tr>
</tbody>
</table>

Remarks  The **EndTime** property has the **Double** type.

See Also  • StartTime (see page 279) property

**FaceName Property**

- Applies To  • **VbaViewChartFont** (see page 158) object
- Description  Gets or sets the text face name string.
- VB Syntax  `object_FACE_NAME [=FaceName]`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>An expression that evaluates to a <strong>VbaViewChartFont</strong> object.</td>
</tr>
<tr>
<td><code>FaceName</code></td>
<td>A <strong>String</strong> value that is the font face name.</td>
</tr>
</tbody>
</table>

Remarks  The **FaceName** property has the **String** type.

**Font Property**

- Applies To  • **VbaViewChartTitle** (see page 159) object
- Description  Gets the chart title font.
**VB Syntax**

```
object.Font
```

**Parameters** | **Definition**
--- | ---
object | An expression that evaluates to a VbaViewChartFont (see page 158) object.

**Remarks**
The `Font` property has the `VbaViewChartFont` (see page 158) object type.

---

**Found Property**

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]**

**Applies To**
- `FindResult` (see page 119) object

**Description**
Gets a `Boolean` value indicating whether the event was found.

**VB Syntax**

```
object.Found
```

**Parameters** | **Definition**
--- | ---
object | An expression that evaluates to a `FindResult` (see page 119) object.

**Remarks**
The `Found` property has the `Boolean` type.

---

**Frame Property**

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]**

**Applies To**
- `Module` (see page 128) object

**Description**
Gets the frame in which the module resides.

**VB Syntax**

```
object.Frame
```

**Parameters** | **Definition**
--- | ---
object | An expression that evaluates to a `Module` (see page 128) object.

**Remarks**
The `Frame` property has the `Frame` (see page 120) object type.

**See Also**
- `Frame` (see page 120) object
- `Frames` (see page 120) object

---

**Frames Property**

**[ Automation Home (see page 3) ] [ Objects (see page 105) ]**
Applies To
• Instrument (see page 121) object

Description
Gets a collection of all logic analyzer frames connected via the multiframe connector.

VB Syntax
object.Frames

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

Remarks
The Frames property has the Frames (see page 120) collection object type. Each item in the collection is a Frame (see page 120) object.

HasLegend Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To
• VbaViewChart (see page 156) object

Description
Gets or sets if the legend is visible.

VB Syntax
object.HasLegend [=HasLegend]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
<tr>
<td>HasLegend</td>
<td>A Boolean value that specifies whether the legend is visible.</td>
</tr>
</tbody>
</table>

Remarks
The HasLegend property has the Boolean type.

HasTitle Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To
• VbaViewChart (see page 156) object
• VbaViewChartAxis (see page 157) object

Description
Gets or sets whether the title is visible.

VB Syntax
object.HasTitle [=HasTitle]
Remarks  The **HasTitle** property has the **Boolean** type.

**Instrument Property**

 Applies To  • **Connect** (see page 118) object

Description  Gets the logic analyzer instrument object.

VB Syntax  

```vbnet
object.Instrument [HostNameOrIpAddress=""]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
<tr>
<td>HasTitle</td>
<td>A <strong>Boolean</strong> value that specifies whether the title is visible.</td>
</tr>
</tbody>
</table>

Remarks  The **Instrument** property has the **Instrument** (see page 121) object type.

**IPAddress Property**

 Applies To  • **Frame** (see page 120) object

Description  Gets a frame's IP address(es). If the frame has more than one LAN card, a comma separated list of IP addresses will be returned.

VB Syntax  

```vbnet
object.IPAddress
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a <strong>Frame</strong> (see page 120) object.</td>
</tr>
<tr>
<td>HostNameOrIpAddress</td>
<td>A <strong>String</strong> that contains the hostname or IP address of the logic analyzer instrument or computer on which the Agilent Logic Analyzer application runs. This parameter is optional—if it is not specified, the computer name specified in the Distributed COM &quot;Location&quot; tab is used (see To verify remote computer Distributed COM properties (see page 39)).</td>
</tr>
</tbody>
</table>

Remarks  The **IPAddress** property has the **String** type.
Item Property

Applies To
- `BusSignalDifferences (see page 109)` object
- `BusSignals (see page 113)` object
- `Frames (see page 120)` object
- `Markers (see page 123)` object
- `Modules (see page 129)` object
- `Probes (see page 135)` object
- `SampleDifferences (see page 149)` object
- `Tools (see page 153)` object
- `Windows (see page 161)` object

Description
Gets one of the objects in a collection given either an index or a name.

VB Syntax

```vbnet
object.Item [IndexOrName]
```
- or-

```vbnet
object IndexOrName
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>An expression that evaluates to one of the objects in the Applies To list above.</td>
</tr>
<tr>
<td>IndexOrName</td>
<td>A <code>Variant</code> that is a <code>Long</code> or <code>String</code> representing the zero-based index in the collection or the name of object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>String Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BusSignalDifferences (see page 109)</code></td>
<td>The <code>BusSignalDifference</code>'s index.</td>
</tr>
<tr>
<td><code>BusSignals (see page 113)</code></td>
<td>The <code>BusSignal</code>'s name (as defined by the <code>Name (see page 269)</code> property).</td>
</tr>
<tr>
<td><code>Frames (see page 120)</code></td>
<td>The Frame's computer name or IP address (as defined by the <code>ComputerName (see page 255)</code> or <code>IPAddress (see page 265)</code> properties, respectively).</td>
</tr>
<tr>
<td><code>Markers (see page 123)</code></td>
<td>The Marker's name (as defined by the <code>Name (see page 269)</code> property).</td>
</tr>
<tr>
<td><code>Modules (see page 129)</code></td>
<td>The Module's name (as defined by the <code>Name (see page 269)</code> property).</td>
</tr>
</tbody>
</table>
Remarks

The **Item** property has the appropriate type of the object in the collection (**BusSignalDifference** (see page 109), **BusSignal** (see page 108), **Frame** (see page 120), **Marker** (see page 123), **Module** (see page 128), **Probe** (see page 135), **SampleDifference** (see page 149), **Tool** (see page 152), or **Window** (see page 160)).

If you specify numbers for `index`, do not store these for later use because the indices might change as items are added or removed.

The **Item** property is the default. Accordingly, you don't have to reference **Item** explicitly, as shown in the syntax.

### Legend Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

**Applies To**

- **VbaViewChart** (see page 156) object

**Description**

Gets the chart legend.

**VB Syntax**

`object.Legend`

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChart (see page 156) object.</td>
</tr>
</tbody>
</table>

**Remarks**

The **Legend** property has the **VbaViewChartLegend** (see page 158) object type.

### Markers Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 124) ]

**Applies To**

- **Instrument** (see page 121) object

**Description**

Gets a collection of all markers.
**VB Syntax**  
object.Markers

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

**Remarks**  
The Markers property has the Markers (see page 123) collection object type. Each item in the collection is a Marker (see page 123) object.

When this property is called, a snapshot of the current markers are returned. If markers are subsequently added or deleted, this property must be called again to get an updated snapshot.

**Model Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**

- Instrument (see page 121) object
- Module (see page 128) object

**Description**

Gets the model number of an object.

**VB Syntax**

object.Model

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Module (see page 128) object.</td>
</tr>
</tbody>
</table>

**Remarks**  
The Model property has the String type.

The following table summarizes the results of using the Model property with the objects in the Applies To list:

<table>
<thead>
<tr>
<th>Object</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument (see page 121)</td>
<td>Gets the instrument’s model number.</td>
</tr>
<tr>
<td>Module (see page 128)</td>
<td>Gets the module’s model number. If the module consists of different card model numbers, they are separated with spaces. For example, an analyzer module would look like “16756A”; a two-card analyzer module containing two different card types would look like “16750A 16750B”.</td>
</tr>
</tbody>
</table>

**Modules Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]
Applies To: Instrument (see page 121) object

Description: Gets a collection of all the enabled hardware modules in the instrument.

VB Syntax: object.Modules

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

Remarks: The Modules property has the Modules (see page 129) collection object type. Each item in the collection is a Module (see page 128) object.

Name Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To: BusSignal (see page 108) object
• BusSignalDifference (see page 109) object
• Marker (see page 123) object
• Module (see page 128) object
• Probe (see page 135) object
• Tool (see page 152) object
• Window (see page 160) object

Description: Gets or sets (where appropriate) the name of an object.

VB Syntax: object.Name [=NewName]

- or -

object

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>An expression that evaluates to one of the objects in the Applies To list above. The Name property is the default property of all objects in the list. Accordingly, you do not have to reference Name explicitly, as shown in the syntax.</td>
</tr>
<tr>
<td>NewName</td>
<td>A String containing the new name of the object.</td>
</tr>
</tbody>
</table>

Remarks: The Name property has the String type.

The following table summarizes the results of using the Name property with some of the objects in the Applies To list:
### NumLines Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

**Applies To**  
- PattgenModule (see page 129) object

**Description**  
Gets the number of lines in the main sequence.

' Get the number of lines in the main sequence.
Dim myNumLines As Long
myNumLines = myPattgenModule.NumLines
MsgBox Str(myNumLines)

**VB Syntax**  
object.NumLines

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a PattgenModule (see page 129) object.</td>
</tr>
</tbody>
</table>

**Remarks**  
The NumLines property has the Long type.

### OccurrencesFound Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**  
- FindResult (see page 119) object

**Description**  
Gets a Long containing the number of times the event was found.

**VB Syntax**  
object.OccurrencesFound
### Remarks
The **OccurrencesFound** property has the **Long** type.

### Options Property

- **Applies To**: CompareWindow (see page 118) object
- **Description**: Gets or sets the Compare window's "XML-format" (in the online help) options specification.

**VB Syntax**

```vbnet
object.Options [=Options]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a CompareWindow (see page 118) object.</td>
</tr>
<tr>
<td>Options</td>
<td>A <strong>String</strong> containing the &quot;XML format&quot; (in the online help)&quot;&lt;Options&gt; element&quot; (in the online help) information.</td>
</tr>
</tbody>
</table>

- **Remarks**: The **Options** property has the **String** type.

### Overview Property

- **Applies To**: Instrument (see page 121) object
- **Description**: Gets an XML version of the Overview (see the "XML format" (in the online help)"<Overview> element" (in the online help)).

' Display the instrument's XML-format overview specification.

```vbnet
Dim myOverview As String
myOverview = myInst.Overview
MsgBox myOverview
```

**VB Syntax**

```vbnet
object.Overview
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>
Remarks The Overview property has the String type.

PanelLocked Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 217) ]

Applies To • Instrument (see page 121) object

Description Indicates whether the instrument's front panel is locked. If locked, it returns the displayed message.

VB Syntax object.PanelLocked Message

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
<tr>
<td>Message</td>
<td>If the font panel is locked, a String containing the displayed message is returned.</td>
</tr>
</tbody>
</table>

Remarks The PanelLocked property has the Boolean type.

Polarity Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To • BusSignal (see page 108) object

Description Gets the polarity of the bus/signal.

VB Syntax object.Polarity

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
</tbody>
</table>

Remarks The Polarity property has the String type.

The polarity is either "+" or "-".

Position Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To • Marker (see page 123) object

Description Gets or sets the marker's time position, in seconds, relative to the trigger.
' Display the marker time position.
Dim myPosition As Double
myPosition = myMarker.Position
MsgBox Str(myPosition)

' Set the marker time position.
myPosition = -30e-9
myMarker.Position = myPosition

**VB Syntax**

```vbnet
object.Position [=Time]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Marker (see page 123) object.</td>
</tr>
<tr>
<td>Time</td>
<td>A <strong>Double</strong> value that is the marker's time position, in seconds, relative to the trigger.</td>
</tr>
</tbody>
</table>

**Remarks**
The **Position** property has the **Double** type.

**Position Property (of VbaViewChartLegend)**

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

**Applies To**

- VbaViewChartLegend (see page 158) object

**Description**
Get or set the chart legend position.

**VB Syntax**

```vbnet
object.Position [=Position]
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartLegend (see page 158) object.</td>
</tr>
<tr>
<td>Position</td>
<td>An <strong>AgtLegendPosition</strong> enumerated type value that can be one of the values described below.</td>
</tr>
</tbody>
</table>

**Remarks**
The **LegendPosition** property can have the following values:

<table>
<thead>
<tr>
<th><strong>AgtLegendPosition</strong></th>
<th><strong>Enum Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AgtLegendPositionBottom</strong></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>AgtLegendPositionLeft</strong></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>AgtLegendPositionRight</strong></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>AgtLegendPositionTop</strong></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
**Probes Property**

Applies To: Instrument (see page 121) object

Description: Gets a collection of all currently defined probes.

VB Syntax: `object.Probes`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

Remarks: The `Probes` property has the `Probes` (see page 135) collection object type. Each item in the collection is a `Probe` (see page 135) object.

When this property is called, a snapshot of the currently defined probes are returned. If probes are subsequently added or deleted, this property must be called again to get an updated snapshot.

**Reference Property**

Applies To: BusSignalDifference (see page 109) object

Description: Gets the reference buffer value associated with the bus/signal difference.

VB Syntax: `object.Reference`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignalDifference (see page 109) object.</td>
</tr>
</tbody>
</table>

Remarks: The `Reference` property has the `String` type.

The value is in base hex, for example: "FF".

**RemoteComputerName Property**

Applies To: Instrument (see page 121) object

Description: Gets or sets the remote computer name.
' Display the remote user computer name.
Dim myRemoteComputerName As String
myRemoteComputerName = myInst.RemoteComputerName
MsgBox Str(myRemoteComputerName)

' Set the remote user computer name.
myRemoteComputerName = "myComputer"
myInst.RemoteComputerName = myRemoteComputerName

**VB Syntax**: `object.RemoteComputerName [=RemoteComputerName]`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a <code>Instrument</code> (see page 121) object.</td>
</tr>
<tr>
<td>RemoteComputerName</td>
<td>A <code>String</code> value that is the name of the remote computer.</td>
</tr>
</tbody>
</table>

**Remarks**: The `RemoteComputerName` property has the `String` type.

**RemoteUserName Property**

`[ Automation Home (see page 3) ] [ Objects (see page 105) ]`

**Applies To**: `Instrument` (see page 121) object

**Description**: Gets or sets the remote user login name.

' Display the remote user login name.
Dim myRemoteUserName As String
myRemoteUserName = myInst.RemoteUserName
MsgBox Str(myRemoteUserName)

' Set the remote user login name.
myRemoteUserName = "myLogin"
myInst.RemoteUserName = myRemoteUserName

**VB Syntax**: `object.RemoteUserName [=RemoteUserName]`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a <code>Instrument</code> (see page 121) object.</td>
</tr>
<tr>
<td>RemoteUserName</td>
<td>A <code>String</code> value that is the login name of the remote user.</td>
</tr>
</tbody>
</table>

**Remarks**: The `RemoteUserName` property has the `String` type.

**RunningStatus Property**

`[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]`

**Applies To**: `Module` (see page 128) object
**Description**

Gets the detailed running status of the module. Call this method when the Status (see page 280) property returns "Running" to get a more detailed running status.

**VB Syntax**

`object.RunningStatus`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Module (see page 128) object.</td>
</tr>
</tbody>
</table>

**Remarks**

The `RunningStatus` property has the `String` type. The string returned is based on the object types defined below.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module (see page 128)</td>
<td>Running · the module is running Stopped · the module has stopped running Initializing · the module is initializing or calibrating Waiting · the module is waiting for an event SelfTest · the instrument is running SelfTest</td>
</tr>
</tbody>
</table>

**See Also**

- Status (see page 280) property

**SampleDifferences Property**

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 109) ]**

**Applies To**

- CompareWindow (see page 118) object

**Description**

Gets a collection of all the samples with differences found in the last comparison.

**VB Syntax**

`object.SampleDifferences`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a CompareWindow (see page 118) object.</td>
</tr>
</tbody>
</table>

**Remarks**

The `SampleDifferences` property has the `SampleDifferences` (see page 149) collection object type. Each item in the collection is a `SampleDifference` (see page 149) object.

When this property is called, a snapshot of the current differences are returned. If another compare is executed, this property must be called again to get an updated snapshot.
**SampleNum Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- SampleDifference (see page 149) object

**Description**
Gets the sample number at which differences occurred.

**VB Syntax**

```vb
object.SampleNum
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a SampleDifferences (see page 149) object.</td>
</tr>
</tbody>
</table>

**Remarks**
The SampleNum property has the Long type.

**SelfTest Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 150) ]

**Applies To**
- Instrument (see page 121) object

**Description**
Gets the SelfTest object.

**VB Syntax**

```vb
object(SelfTest
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

**Remarks**
The SelfTest property has the SelfTest (see page 149) object type.

**See Also**
- SelfTest (see page 149) object

**Setup Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 58) ]

**Applies To**
- AnalyzerModule (see page 106) object

**Description**
Gets or sets the logic analyzer's "XML-format" (in the online help) setup specification.

```vb
' Display the logic analyzer setup specification.
Dim mySetup As String
mySetup = myInst.GetModuleByName("My 1690A-1").Setup
MsgBox mySetup
```
' Set the logic analyzer setup specification.
myInst.GetModuleByName("My 1690A-1").Setup = mySetup

**VB Syntax**

```vbnet
object.Setup [=XMLSetupSpec]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an AnalyzerModule (see page 106) object.</td>
</tr>
<tr>
<td>XMLSetupSpec</td>
<td>A <strong>String</strong> containing the &quot;XML format&quot; (in the online help)&quot;&lt;Module&gt; element&quot; (in the online help) information.</td>
</tr>
</tbody>
</table>

**Remarks**
The *Setup* property has the **String** type.

**Size Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**Applies To**
- **VbaViewChartFont** (see page 158) object

**Description**
Gets or sets the text size.

**VB Syntax**

```vbnet
object.Size [=Size]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewChartFont (see page 158) object.</td>
</tr>
<tr>
<td>Size</td>
<td>A <strong>Long</strong> value that specifies the text size in points.</td>
</tr>
</tbody>
</table>

**Remarks**
The *Size* property has the **Long** type.

**Slot Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- **Module** (see page 128) object

**Description**
Gets the module's slot location in the frame (see the *Frame* (see page 263) property). If the module is comprised of a single card, the letter corresponding to the slot is returned. Possible values are "A" through "F". If the module is comprised of a multi-card set, a string identifying all slots occupied by the module, highlighting the master card slot, is returned. Format: "<starting slot>-<ending slot>{m=<master slot>}". Example: "A-C[m=B]".

**VB Syntax**

```vbnet
object.Slot
```
### Remarks
The **Slot** property has the **String** type.

### See Also
- **Frame** (see page 120) object
- **Frames** (see page 120) object

### StartSample Property

#### Parameters Definition
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Module (see page 128) object.</td>
</tr>
</tbody>
</table>

#### Remarks
The **StartSample** property has the **Long** type.

The starting sample number is relative to trigger; therefore, a starting sample number equal to -1024 means the trigger is at sample 0, 1024 samples after the starting sample.

#### See Also
- **EndSample** (see page 261) property

### StartTime Property

#### Parameters Definition
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an SampleBusSignalData (see page 138) object.</td>
</tr>
</tbody>
</table>

#### Remarks
The **StartTime** property has the **Double** type.
The starting time is relative to trigger and can therefore be a negative number.

**See Also**
- EndTime (see page 262) property

### Status Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

**Applies To**
- Instrument (see page 121) object
- Module (see page 128) object

**Description**
Gets the run status. The Instrument object's **Status** property returns the run status of all data acquisition modules (that is, not pattern generator modules), tools, and windows. The Module object's **Status** property only returns the run status of that specific module.

**NOTE**
Instead of polling the **Status** property in a loop, use the object’s **WaitComplete** (see page 238) method to wait for a measurement to complete.

When a module is running repetitively, either "Stopped" or "Running" can be returned based on the current state of the module. Use the Instrument object’s **Status** property which will always return "Running" during a repetitive run.

**VB Syntax**

```
object.Status
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
</tbody>
</table>

**Remarks**
The **Status** property has the **String** type. The string returned is based on the object types defined below.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument (see page 121)</td>
<td>Running - at least one of the data acquisition modules, tools, or windows is running</td>
</tr>
<tr>
<td></td>
<td>Stopped - data acquisition modules, tools, or windows have stopped running</td>
</tr>
<tr>
<td>Module (see page 128)</td>
<td>Running - the module is running</td>
</tr>
<tr>
<td></td>
<td>Stopped - the module has stopped running</td>
</tr>
</tbody>
</table>

**See Also**
- RunningStatus (see page 275) property
StatusMsg Property

Applies To
- Module (see page 128) object

Description
Gets the module's verbose status message.

VB Syntax
object.StatusMsg

Parameters | Definition
---|---
object | An expression that evaluates to a Module (see page 128) object.

Return Value
A String containing the verbose status message.

NOTE
This message is not guaranteed to be static and, therefore, should not be parsed. Use only for display purposes.

If you want to know the specific status of a module, see the RunningStatus (see page 275) or Status (see page 280) properties.

See Also
- RunningStatus (see page 275) property
- Status (see page 280) property

SubrowFound Property

Applies To
- FindResult (see page 119) object

Description
Gets a Long containing the subrow number if the data sample contains subrows.

VB Syntax
object.SubrowFound

Parameters | Definition
---|---
object | An expression that evaluates to a FindResult (see page 119) object.

Remarks
The SubrowFound property has the Long type.

Symbols Property

Applies To
- BusSignal (see page 108) object
Description  
Gets or sets the symbols associated with a bus/signal.

' Display a bus/signal's symbols.
Dim mySymbols As String
mySymbols = _
    myInst.GetModuleByName("My 1690A-1").BusSignals("My Bus 1").Symbols
MsgBox mySymbols

' Set a bus/signal's symbols.
mySymbols = "<Symbols><Symbol Name='My Symbol' Operator='Equals' _ + _
    "Value='hFF'/></Symbols>"
myInst.GetModuleByName("My 1690A-1").BusSignals("My Bus 1").Symbols = _
    mySymbols

VB Syntax  
object.Symbols [=XMLSymbols]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignal (see page 108) object.</td>
</tr>
<tr>
<td>XMLSymbols</td>
<td>A String containing the &quot;XML format&quot; (in the online help)&quot;&lt;Symbols&gt; element&quot; (in the online help) information.</td>
</tr>
</tbody>
</table>

Remarks  
The Symbols property has the String type.

TargetControlPort Property

[ Automation Home (see page 3) ] [ Objects (see page 105) ]

Applies To  
- Frame (see page 120) object

Description  
Gets or sets the target control port value. The port is 8-bit TTL and can be used to remotely control switches on your device under test.

VB Syntax  
object.TargetControlPort [=Value]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a Frame (see page 120) object.</td>
</tr>
</tbody>
</table>

Value  
A String containing a port value. The value returned is the value present on the physical pins. The value can be set to a decimal, hexadecimal, octal, or binary number with optional don’t care digits "x". The don’t care digit is used to indicate that the value will stay the same (don’t care). To specify a number base, use the following prefixes:
- h - for hexadecimal
- b - binary
- o - octal
- d - decimal
For example: "hf", "b11110000", "bxxxx1xxx".
Remarks The TargetControlPort property has the String type.

TextColor Property

Applies To • Marker (see page 123) object

Description Gets or sets the marker text color.

' Display the marker text color.
Dim myTextColor As Long
myTextColor = myMarker.TextColor
MsgBox Str(myTextColor)

' Set the marker text color to red.
myTextColor = &H000000FF
myMarker.TextColor = myTextColor

VB Syntax object.TextColor [=Color]

Parameters Definition
object An expression that evaluates to a Marker (see page 123) object.
Color A Long value that represents the marker text color.

Remarks The TextColor property has the Long type.

Color values have the following hexadecimal form: 0x00BBGGRR. The low-order byte (RR) contains a value for the relative intensity of red; the second byte (GG) contains a value for green; and the third byte (BB) contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is &HFF. The color white is &H00FFFFFF, black is &H00000000, and red is &H000000FF.

TimeFound Property

Applies To • FindResult (see page 119) object

Description Gets a Double containing the time (in seconds) the event occurred in the data. You can call the GetDataByTime (see page 192) method with this value to get more details.

VB Syntax object.TimeFound

Parameters Definition
object An expression that evaluates to a FindResult (see page 119) object.
Remarks  The **TimeFound** property has the **Double** type.

**TimeFoundString Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]

Applies To  •  **FindResult** (see page 119) object

Description  Gets the time found as a string.

**VB Syntax**  

```vbnet
object.TimeFoundString
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a FindResult (see page 119) object.</td>
</tr>
</tbody>
</table>

Remarks  The **TimeFoundString** property has the **String** type.

**Title Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

Applies To  •  **VbaViewChart** (see page 156) object

•  **VbaViewChartAxis** (see page 157) object

Description  Gets the title of the chart or axis.

**VB Syntax**  

```vbnet
object.Title
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
</tbody>
</table>

Remarks  The **Title** property has the **VbaViewChartTitle** (see page 159) object type.

**Tools Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 153) ]

Applies To  •  **Instrument** (see page 121) object

Description  Gets a collection of all active software tools.

**VB Syntax**  

```vbnet
object.Tools
```
The **Tools** property has the **Tools** (see page 153) collection object type. Each item in the collection is a **Tool** (see page 152) object.

When this property is called, a snapshot of the currently active tools are returned. If tools are subsequently added or deleted from the Overview, this property must be called again to get an updated snapshot.

### Remarks

The **Tools** property has the **Tools** (see page 153) collection object type. Each item in the collection is a **Tool** (see page 152) object.

When this property is called, a snapshot of the currently active tools are returned. If tools are subsequently added or deleted from the Overview, this property must be called again to get an updated snapshot.

### Trigger Property

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 54) ]**

**Applies To**

- **AnalyzerModule** (see page 106) object

**Description**

Gets or sets the logic analyzer's "XML-format" (in the online help) trigger specification.

```vbnet
' Display the logic analyzer trigger specification.
Dim myTrigger As String
myTrigger = myInst.GetModuleByName("My 1690A-1").Trigger
MsgBox myTrigger

' Set the logic analyzer trigger specification.
myInst.GetModuleByName("My 1690A-1").Trigger = myTrigger
```

**VB Syntax**

```vbnet
object.Trigger [=XMLTriggerSpec]
```

### Remarks

The **Trigger** property has the **String** type.

### Type Property

**[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example ]**

**Applies To**

- **BusSignalData** (see page 108) object
- **Module** (see page 128) object
- **Tool** (see page 152) object
- **Window (see page 160)** object

**Description**

Gets the BusSignalData (see page 108), Module (see page 128), Tool (see page 152), or Window (see page 160) object's type. The type can be used to identify sub-objects and their specific methods and properties.

**VB Syntax**

`object.Type`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to one of the objects in the &quot;Applies to&quot; list above.</td>
</tr>
</tbody>
</table>

**Remarks**

The **Type** property has the **String** type.

The following tables show how types correspond to objects.

<table>
<thead>
<tr>
<th>BusSignalData Type</th>
<th>BusSignalData Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>SampleBusSignalData (see page 138)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Module Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzer</td>
<td>AnalyzerModule (see page 106)</td>
</tr>
<tr>
<td>ExternalScope</td>
<td>Module (see page 128)</td>
</tr>
<tr>
<td>Import</td>
<td>Module (see page 128)</td>
</tr>
<tr>
<td>Pattgen</td>
<td>PattgenModule (see page 129)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window Type</th>
<th>Window Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>CompareWindow (see page 118)</td>
</tr>
</tbody>
</table>

**Value Property**

- **BusSignalDifference (see page 109)** object

**Description**

Gets the data value associated with the bus/signal difference.

**VB Syntax**

`object.Value`
The **Value** property has the **String** type.

The value is in base hex, for example: "FF".

### VBAVersion Property

- **Applies To**: Instrument (see page 121) object
- **Description**: Gets the version number of VBA.
- **VB Syntax**: `object.VBAVersion`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a BusSignalDifference (see page 109) object.</td>
</tr>
</tbody>
</table>

**Remarks**: The **VBAVersion** property has the **String** type.

### VBE Property

- **Applies To**: Instrument (see page 121) object
- **Description**: Gets the VBE extensibility object.
- **VB Syntax**: `object.VBE`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

**Remarks**: The **VBE** property has the **VBE** object type.

### Version Property

- **Applies To**: Instrument (see page 121) object
Description  Gets the version number of the system software.

**VB Syntax**  
`object.Version`  

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

**Remarks**  
The `Version` property has the `String` type.

The format of the version string is `##.##.####` (for example, "02.00.0000")

**WebBrowser Property (for VbaViewWindow object)**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 160) ]

**Applies To**  
• VbaViewWindow (see page 160) object

**Description**  
Gets the Web Browser view.

**VB Syntax**  
`object.WebBrowser`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWindow (see page 160) object.</td>
</tr>
</tbody>
</table>

**Remarks**  
The `WebBrowser` property has the `VbaViewWebBrowser` (see page 159) object type.

**Requirements**  
• Version (see page 62): 3.20 or later.

**WebBrowser Property (for VbaViewWebBrowser object)**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 159) ]

**Applies To**  
• VbaViewWebBrowser (see page 159) object

**Description**  
Gets the contained IWebBrowser2 interface.

**VB Syntax**  
`object.WebBrowser`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to a VbaViewWebBrowser (see page 159) object.</td>
</tr>
</tbody>
</table>
Remarks  The **WebBrowser** property returns the interface to the contained Internet Explorer Control called IWebBrowser2. In Visual Basic, this can only be used if you create a reference to "Microsoft Internet Controls".

Requirements  •  **Version (see page 62):** 3.20 or later.

**Windows Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 162) ]

**Applies To**  •  **Instrument (see page 121) object**

**Description**  Gets a collection of all windows/viewers.

**VB Syntax**  `object.Windows`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An expression that evaluates to an Instrument (see page 121) object.</td>
</tr>
</tbody>
</table>

Remarks  The **Windows** property has the **Windows (see page 161) collection object type**. Each item in the collection is a **Window (see page 160) object**.

When this property is called, a snapshot of the currently active windows are returned. If windows are subsequently added or deleted from the Overview, this property must be called again to get an updated snapshot.

**_NewEnum Property**

[ Automation Home (see page 3) ] [ Objects (see page 105) ] [ Example (see page 290) ]

**Applies To**  •  **BusSignalDifferences (see page 109) object**

•  **BusSignals (see page 113) object**

•  **Frames (see page 120) object**

•  **Markers (see page 123) object**

•  **Modules (see page 129) object**

•  **Probes (see page 135) object**

•  **SampleDifferences (see page 149) object**

•  **Tools (see page 153) object**

•  **Windows (see page 161) object**

**Description**  References objects in a collection.

**VB Syntax**  `object._NewEnum`
The **_NewEnum** property has the appropriate type of the object in the collection (**BusSignalDifference** (see page 109), **BusSignal** (see page 108), **Frame** (see page 120), **Marker** (see page 123), **Module** (see page 128), **SampleDifference** (see page 149), **Tool** (see page 152), or **Window** (see page 160)).

With Visual C++, you can browse a collection to find a particular item by using the **_NewEnum** property or the **Item** (see page 266) property. In Visual Basic, you do not need to use the **_NewEnum** property because it is automatically used in the implementation of **For Each** ... **Next**.

**_NewEnum Example**

This Visual Basic example uses the **_NewEnum** property to iterate through all buses/signals and display their names.

**Visual Basic**

```vbnet
' Display all of the bus/signal names.
Dim myBusSignals As AgtLA.BusSignals
Set myBusSignals = myInst.GetModuleByName("My 1690A-1").BusSignals

Dim myBusSignalNames As String
Dim myBusSignal As AgtLA.BusSignal

For Each myBusSignal in myBusSignals
    ' Add the bus/signal name to the string.
    myBusSignalNames = myBusSignalNames + vbCrLf + myBusSignal.Name
Next
MsgBox "Bus/signal names: " + myBusSignalNames
```

**Visual C++**

```cpp
// This simple Visual C++ Console application demonstrates how to
// use the Agilent 168x/169x/169xx COM interface to iterate through
// all buses/signals and display their names.
//
// This project was created in Visual C++ Developer. To create a
// similar project:
//
// - Execute File -> New
// - Select the Projects tab
// - Select "Win32 Console Application"
// - Select A "hello,World!" application (Visual Studio 6.0)
//
// To make this buildable, you need to specify your "import" path
// in stdafx.h (search for "TODO" in that file). For example, add:
// #import "C:/Program Files/Agilent Technologies/Logic Analyzer/LA 
```
COM Automation Reference 4

// COM Automation/agClientSvr.dll
//
// To run, you need to specify the host logic analyzer to connect
// to (search for "TODO" below).
//
#include "stdafx.h"

//////////////////////////////////////////////////////////////////////
// Forward declarations.
//
void DisplayError(_com_error& err);

//////////////////////////////////////////////////////////////////////
// main() entry point.
//
int main(int argc, char* argv[]) {
  printf("*** Main()\n");

  // Initialize the Microsoft COM/ActiveX library.
  HRESULT hr = CoInitialize(0);
  if (SUCCEEDED(hr)) {
    try { // Catch any unexpected run-time errors.
      _bstr_t hostname = "mtx33"; // TODO, use your logic
      // analysis system hostname.
      printf("Connecting to instrument '%s'\n", (char*) hostname);

      // Create the connect object and get the instrument object.
      AgtLA::IConnectPtr pConnect =
        AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));
      AgtLA::IInstrumentPtr pInst =
        pConnect->GetInstrument(hostname);

      // Load the configuration file.
      _bstr_t configFile = "C:\LA\Configs\config.ala";
      printf("Loading the config file '%s'\n", (char*) configFile);
      pInst->Open(configFile, FALSE, "", TRUE);

      // Display all of the bus/signal names.
      _bstr_t moduleName = "MPC860 Demo Board";
      _bstr_t busSignal;
      AgtLA::IAnalyzerModulePtr pAnalyzer =
        pInst->GetModuleByName(moduleName);
      AgtLA::IBusSignalsPtr pBusSignals = pAnalyzer->GetBusSignals();
      for (long i = 0; i < pBusSignals->GetCount(); i++)
      {
busSignal = pBusSignals->GetItem(i)->GetName();
printf("Bus/signal name '%s'.\n", (char*) busSignal);
}
}  
catch (_com_error& e) {
    DisplayError(e);
}

// Uninitialize the Microsoft COM/ActiveX library.
CoUninitialize();
}
else {
    printf("CoInitialize failed\n");
}

return 0;
}

/////////////////////////////////////////////////////////////////////

// Displays the last error -- used to show the last exception
// information.
//
// void DisplayError(_com_error& error)
{
    printf("*** DisplayError()\n");
    printf("Fatal Unexpected Error:\n");
    printf(" Error Number = %08lx\n", error.Error());

    static char errorStr[1024];
    _bstr_t desc = error.Description();

    if (desc.length() == 0) {
        // Don't have a description string.
        strcpy(errorStr, error.ErrorMessage());
        int nLen = lstrlen(errorStr);
        // Remove funny carriage return ctrl<M>.
        if (nLen > 2 && (errorStr[nLen - 2] == 0xd)) {
            errorStr[nLen - 2] = '\0';
        }
    } else {
        strcpy(errorStr, desc);
    }

    printf(" Error Message = %s\n", (char*) errorStr);"
5
What's Changed

- The BusSignals (see page 113) object no longer returns "Time" data as one of the buses/signals in the collection. Now, to get the time associated with bus/signal data, use the GetTime (see page 204) method of the SampleBusSignalData (see page 138) object.
### Index

#### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NewEnum example</td>
<td>290</td>
</tr>
<tr>
<td>_NewEnum property</td>
<td>289</td>
</tr>
</tbody>
</table>

#### Numerics

<table>
<thead>
<tr>
<th>Numerical</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16900 service</td>
<td>20</td>
</tr>
</tbody>
</table>

#### A

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity property</td>
<td>245</td>
</tr>
<tr>
<td>Add method, BusSignals object</td>
<td>168</td>
</tr>
<tr>
<td>Add method, Markers object</td>
<td>169</td>
</tr>
<tr>
<td>AddPointArrays method</td>
<td>170</td>
</tr>
<tr>
<td>AddXML method</td>
<td>170</td>
</tr>
<tr>
<td>advanced triggers, COM automation</td>
<td>54</td>
</tr>
<tr>
<td>agClientSvr.tlh header file, Visual C++</td>
<td>70</td>
</tr>
<tr>
<td>agFirewSP2.wsf</td>
<td>18</td>
</tr>
<tr>
<td>Agilent Logic Analysis Application program</td>
<td>18</td>
</tr>
<tr>
<td>Agilent Logic Analysis Service program</td>
<td>18</td>
</tr>
<tr>
<td>AgtDataType</td>
<td>190</td>
</tr>
<tr>
<td>AnalyzerModule object</td>
<td>106</td>
</tr>
<tr>
<td>automation overview</td>
<td>13</td>
</tr>
<tr>
<td>Axis property</td>
<td>246</td>
</tr>
<tr>
<td>AxisBase property</td>
<td>246</td>
</tr>
</tbody>
</table>

#### B

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroudColor property</td>
<td>247</td>
</tr>
<tr>
<td>BitSize property</td>
<td>248</td>
</tr>
<tr>
<td>BitSize property (of VbaViewChartAxis object)</td>
<td>248</td>
</tr>
<tr>
<td>Bold property</td>
<td>248</td>
</tr>
<tr>
<td>BusSignal object</td>
<td>108</td>
</tr>
<tr>
<td>BusSignalData object</td>
<td>108</td>
</tr>
<tr>
<td>BusSignalData property</td>
<td>249</td>
</tr>
<tr>
<td>BusSignalDifference property</td>
<td>109</td>
</tr>
<tr>
<td>BusSignalDifferences example</td>
<td>109</td>
</tr>
<tr>
<td>BusSignalDifferences object</td>
<td>109</td>
</tr>
<tr>
<td>BusSignalDifferences property</td>
<td>250</td>
</tr>
<tr>
<td>BusSignals example</td>
<td>114</td>
</tr>
<tr>
<td>BusSignals object</td>
<td>113</td>
</tr>
<tr>
<td>BusSignals property</td>
<td>250</td>
</tr>
<tr>
<td>BusSignalType property</td>
<td>249</td>
</tr>
<tr>
<td>ByteSize property</td>
<td>251</td>
</tr>
</tbody>
</table>

#### C

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption property</td>
<td>251</td>
</tr>
<tr>
<td>CardModels property</td>
<td>252</td>
</tr>
<tr>
<td>changes since previous releases</td>
<td>293</td>
</tr>
</tbody>
</table>

#### D

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels property</td>
<td>252</td>
</tr>
<tr>
<td>Chart property</td>
<td>253</td>
</tr>
<tr>
<td>ChartType property</td>
<td>253</td>
</tr>
<tr>
<td>Clear method</td>
<td>171</td>
</tr>
<tr>
<td>ClearOutput method</td>
<td>172</td>
</tr>
<tr>
<td>Close method</td>
<td>172</td>
</tr>
<tr>
<td>Color property</td>
<td>254</td>
</tr>
<tr>
<td>COM automation</td>
<td>3</td>
</tr>
<tr>
<td>COM automation, firewall settings</td>
<td>16</td>
</tr>
<tr>
<td>COM version checking</td>
<td>62</td>
</tr>
<tr>
<td>Comments property</td>
<td>255</td>
</tr>
<tr>
<td>CompareWindow object</td>
<td>118</td>
</tr>
<tr>
<td>ComputerName property</td>
<td>255</td>
</tr>
<tr>
<td>Connect method</td>
<td>172</td>
</tr>
<tr>
<td>Connect object</td>
<td>118</td>
</tr>
<tr>
<td>connect, remote, Windows XP firewall settings</td>
<td>20</td>
</tr>
<tr>
<td>ConnectSystem object</td>
<td>119</td>
</tr>
<tr>
<td>CopyFile method</td>
<td>173</td>
</tr>
<tr>
<td>Count example</td>
<td>256</td>
</tr>
<tr>
<td>Count property</td>
<td>256</td>
</tr>
<tr>
<td>CreatorName property</td>
<td>259</td>
</tr>
</tbody>
</table>

#### E

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndSample property</td>
<td>261</td>
</tr>
<tr>
<td>EndTime property</td>
<td>262</td>
</tr>
<tr>
<td>events in Find and SimpleTrigger methods</td>
<td>233</td>
</tr>
<tr>
<td>examples, LabVIEW</td>
<td>75</td>
</tr>
<tr>
<td>examples, Visual Basic programs</td>
<td>46</td>
</tr>
<tr>
<td>Excel VB macro example</td>
<td>42</td>
</tr>
<tr>
<td>Execute method</td>
<td>178</td>
</tr>
<tr>
<td>Export method</td>
<td>179</td>
</tr>
<tr>
<td>ExportEx method</td>
<td>180</td>
</tr>
</tbody>
</table>

#### F

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FaceName property</td>
<td>262</td>
</tr>
<tr>
<td>File and Printer Sharing</td>
<td>18</td>
</tr>
<tr>
<td>Find example</td>
<td>185</td>
</tr>
<tr>
<td>Find method</td>
<td>184</td>
</tr>
<tr>
<td>FindNext method</td>
<td>188</td>
</tr>
<tr>
<td>FindPrev method</td>
<td>189</td>
</tr>
<tr>
<td>FindResult object</td>
<td>119</td>
</tr>
<tr>
<td>firewall, settings</td>
<td>16</td>
</tr>
<tr>
<td>Firewall, Windows</td>
<td>16, 18</td>
</tr>
<tr>
<td>folders, shared, Windows XP firewall settings</td>
<td>20</td>
</tr>
<tr>
<td>Font property</td>
<td>262</td>
</tr>
<tr>
<td>Found property</td>
<td>263</td>
</tr>
<tr>
<td>Frame object</td>
<td>120</td>
</tr>
<tr>
<td>Frame property</td>
<td>263</td>
</tr>
<tr>
<td>Frames object</td>
<td>120</td>
</tr>
<tr>
<td>Frames property</td>
<td>263</td>
</tr>
</tbody>
</table>

#### G

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get/Put methods, Visual C++</td>
<td>70</td>
</tr>
<tr>
<td>GetDataBySample method</td>
<td>189</td>
</tr>
<tr>
<td>GetDataByTime method</td>
<td>192</td>
</tr>
<tr>
<td>GetGroupCaption method</td>
<td>193</td>
</tr>
<tr>
<td>GetInstrument method, Visual C++</td>
<td>70</td>
</tr>
<tr>
<td>GetLine method</td>
<td>194</td>
</tr>
<tr>
<td>GetLineLabel method</td>
<td>196</td>
</tr>
<tr>
<td>GetModuleByName example</td>
<td>197</td>
</tr>
<tr>
<td>GetModuleByName method</td>
<td>196</td>
</tr>
<tr>
<td>GetNumSamples method</td>
<td>199</td>
</tr>
<tr>
<td>GetProbeByName method</td>
<td>200</td>
</tr>
<tr>
<td>GetRawData method</td>
<td>200</td>
</tr>
<tr>
<td>GetRawTimingZoomData method</td>
<td>202</td>
</tr>
<tr>
<td>GetRemoteInfo method</td>
<td>203</td>
</tr>
<tr>
<td>GetSampleNumByTime method</td>
<td>204</td>
</tr>
<tr>
<td>GetTime method</td>
<td>204</td>
</tr>
<tr>
<td>GetToolByName method</td>
<td>205</td>
</tr>
<tr>
<td>GetValueCaption method</td>
<td>206</td>
</tr>
<tr>
<td>GetWindowByName method</td>
<td>206</td>
</tr>
<tr>
<td>GoOffline example</td>
<td>207</td>
</tr>
<tr>
<td>GoOffline method</td>
<td>207</td>
</tr>
<tr>
<td>GoOnline method</td>
<td>211</td>
</tr>
<tr>
<td>GoToPosition method</td>
<td>212</td>
</tr>
</tbody>
</table>
Index

H
HasLegend property, 264
HasTitle property, 264

I
ICMP settings, 18, 20
ICMv4 incoming echo requests, 32
Import method, 212
ImportEx method, 213
inbound rules, Windows Firewall with Advanced Security (Vista), 26
InsertLine method, 214
installing COM automation client software, 34
InstructionOrVector string, 194
Instrument object, 121
Instrument property, 265
Internet Control Message Protocol settings, 18, 20
IPAddress property, 265
IsOnline method, 214
IsTimingZoom method, 215
Item property, 266

L
LabVIEW, 72
LabVIEW examples, 75
LabVIEW tutorial, 72
Legend property, 267
loading configurations, COM automation, 46
local area connection properties, 20

M
macro, pattern generator, 194
macro, VBA, 42
Marker object, 123
Markers example, 124
Markers object, 123
Markers property, 267
methods quick reference, COM automation, 90
methods, COM automation, 166
Microsoft RPC endport mapper service, 18, 20
Model property, 268
Module object, 128
Modules object, 129
Modules property, 268
multiframe configurations, Windows XP firewall settings, 20

N
Name property, 269
netbios-ds service, 20
netbios-ss service, 20
NetOp, Windows XP firewall settings, 16, 20
network connections, 20
networking configurations supported for COM automation, 33
New method, 215
notices, 2
NumLines property, 270

O
object hierarchy overview, 102
object quick reference, 105
objects quick reference, COM automation, 90
OccurrencesFound property, 270
Open method, 216
Options property, 271
Overview property, 271

P
PanelLock example, 217
PanelLock method, 216
PanelLocked property, 272
PanelUnlock method, 220
PattgenModule example, 131
PattgenModule object, 129
Perl, 76
Polarity property, 272
ports, network, firewall access to, 16
Position property, 272
Position property (of VbaViewChartLegend object), 273
Probe object, 135
Probes example, 136
Probes object, 135
Probes property, 274
programs, firewall access to, 16
programs, Visual Basic, 46
properties quick reference, COM automation, 90
properties, COM automation, 243
properties, local area connection, 20
Put/Get methods, Visual C++, 70
Python, 80
QueryCommand method, 220
range (data), specifying, 180
RealVNC, Windows XP firewall settings, 16, 20
RecallTriggerByName method, 224
RecallTriggerByFile method, 223
Reference method, 224
Reference property, 274
reference, COM automation, 89
remote connection, Windows XP firewall settings, 20
Remote Desktop, Windows XP firewall settings, 18, 20
RemoteComputerName property, 274
RemoteUserName property, 275
Remove method, BusSignals object, 224
Remove method, Markers object, 225
RemoveAll method, 225
RemoveLine method, 226
RemoveXML method, 225
Reset method, 226
Resume method, 227
return values and DataTypes, 190
Run method, 227
Run method (PattgenModule object), 228
running measurements, COM automation, 46
RunningStatus property, 275
sample numbers, specifying a data range by, 180
SampleBusSignalData example, 139
SampleBusSignalData object, 138
SampleDifference object, 143
SampleDifferences object, 149
SampleDifferences property, 276
SampleNum property, 277
sampling mode, changing, 58
Save method, 228
SelfTest example, 150
SelfTest object, 149
SelfTest property, 277
SendFile method, 229
Service Pack 2 (SP2), Windows XP Professional, 16
SetGroupCaption method, 229
SetLine method, 230
SetLineLabel method, 230
setting up for COM automation, 15
Setup property, 277
SetValue method, 231
SetValueArray method, 231
SetValueCaption method, 232
shared folders, Windows XP firewall settings, 20
simple file sharing, Workgroup DCOM configuration, 37
simple triggers, COM automation, 51
SimpleTrigger method, 232
Size property, 278
Slot property, 278
SP2 (Service Pack 2), Windows XP Professional, 16
StartSample property, 279
StartTime property, 279
Status property, 280
StatusMsg property, 281
Step method, 235
Stop method (Instrument object), 236
Stop method (PattgenModule object), 236
storing captured data, COM automation, 46
SubrowFound property, 281
Symbols property, 281
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetControlPort property</td>
<td>282</td>
</tr>
<tr>
<td>Tcl</td>
<td>84</td>
</tr>
<tr>
<td>TestAll method</td>
<td>237</td>
</tr>
<tr>
<td>testing distributed COM connections</td>
<td>35</td>
</tr>
<tr>
<td>TextColor property</td>
<td>283</td>
</tr>
<tr>
<td>time, specifying a data range by</td>
<td>180</td>
</tr>
<tr>
<td>TimeFound property</td>
<td>283</td>
</tr>
<tr>
<td>TimeFoundString property</td>
<td>284</td>
</tr>
<tr>
<td>Title property</td>
<td>284</td>
</tr>
<tr>
<td>Tool object</td>
<td>152</td>
</tr>
<tr>
<td>Tools example</td>
<td>153</td>
</tr>
<tr>
<td>Tools object</td>
<td>153</td>
</tr>
<tr>
<td>Tools property</td>
<td>284</td>
</tr>
<tr>
<td>trademarks</td>
<td>2</td>
</tr>
<tr>
<td>Trigger property</td>
<td>285</td>
</tr>
<tr>
<td>troubleshooting, Distributed COM</td>
<td>36</td>
</tr>
<tr>
<td>tutorial, LabVIEW</td>
<td>72</td>
</tr>
<tr>
<td>type library, importing</td>
<td>42, 45</td>
</tr>
<tr>
<td>Type property</td>
<td>285</td>
</tr>
<tr>
<td>using COM automation</td>
<td>41</td>
</tr>
<tr>
<td>Value property</td>
<td>286</td>
</tr>
<tr>
<td>VBAAddHlepTopic method</td>
<td>237</td>
</tr>
<tr>
<td>VBARunMacro method</td>
<td>237</td>
</tr>
<tr>
<td>VBARunRPICommand method</td>
<td>238</td>
</tr>
<tr>
<td>VBAVersion property</td>
<td>287</td>
</tr>
<tr>
<td>VbaViewChart object</td>
<td>156</td>
</tr>
<tr>
<td>VbaViewChartAxis object</td>
<td>157</td>
</tr>
<tr>
<td>VbaViewChartData object</td>
<td>157</td>
</tr>
<tr>
<td>VbaViewChartFont object</td>
<td>158</td>
</tr>
<tr>
<td>VbaViewChartLegend object</td>
<td>158</td>
</tr>
<tr>
<td>VbaViewChartTitle object</td>
<td>159</td>
</tr>
<tr>
<td>VbaViewChartTitle object</td>
<td>159</td>
</tr>
<tr>
<td>VbaViewChartTitle object</td>
<td>159</td>
</tr>
<tr>
<td>VbaViewChart object</td>
<td>159</td>
</tr>
<tr>
<td>VbaViewWebBrowser example</td>
<td>159</td>
</tr>
<tr>
<td>VbaViewWindow example</td>
<td>160</td>
</tr>
<tr>
<td>VbaViewWindow object</td>
<td>160</td>
</tr>
<tr>
<td>VBE property</td>
<td>287</td>
</tr>
<tr>
<td>version checking, COM</td>
<td>62</td>
</tr>
<tr>
<td>Version property</td>
<td>287</td>
</tr>
<tr>
<td>Vista, firewall settings for DCOM</td>
<td>25</td>
</tr>
<tr>
<td>Visual Basic (in Visual Studio)</td>
<td>45</td>
</tr>
<tr>
<td>Visual Basic examples</td>
<td>69</td>
</tr>
<tr>
<td>Visual Basic program examples</td>
<td>46</td>
</tr>
<tr>
<td>Visual C++</td>
<td>70</td>
</tr>
<tr>
<td>VNC, Windows XP firewall settings</td>
<td>16, 20</td>
</tr>
<tr>
<td>WaitComplete example</td>
<td>239</td>
</tr>
<tr>
<td>WaitComplete method</td>
<td>238</td>
</tr>
<tr>
<td>web server, Windows XP firewall settings</td>
<td>18, 20</td>
</tr>
<tr>
<td>WebBrowser property</td>
<td>288</td>
</tr>
<tr>
<td>what’s changed</td>
<td>293</td>
</tr>
</tbody>
</table>