

**COM Automation**

**Online Help**



**Agilent Technologies**

# 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.

## 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.

## Manual Part Number

Version 03.83.0000

## Edition

October 12, 2009

Available in electronic format only

Agilent Technologies, Inc.  
1900 Garden of the Gods Road  
Colorado Springs, CO 80907 USA

## 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

Using Python [80](#)

Using Tcl [84](#)

#### **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



ImportEx Method [213](#)  
 InsertLine Method [214](#)  
 IsOnline Method [214](#)  
 IsTimingZoom Method [215](#)  
 New Method [215](#)  
 Open Method [216](#)  
 PanelLock Method [216](#)  
 PanelUnlock Method [220](#)  
 QueryCommand Method [220](#)  
 RecallTriggerByFile Method [223](#)  
 RecallTriggerByName Method [224](#)  
 RecvFile Method [224](#)  
 Remove Method (BusSignals Object) [224](#)  
 Remove Method (Markers Object) [225](#)  
 RemoveAll Method [225](#)  
 RemoveXML Method [225](#)  
 RemoveLine Method [226](#)  
 Reset Method (PattgenModule Object) [226](#)  
 Resume Method (PattgenModule Object) [227](#)  
 Run Method (Instrument Object) [227](#)  
 Run Method (PattgenModule Object) [228](#)  
 Save Method [228](#)  
 SendFile Method [229](#)  
 SetGroupCaption Method [229](#)  
 SetLine Method [230](#)  
 SetLineLabel Method [230](#)  
 SetValue Method [231](#)  
 SetValueArray Method [231](#)  
 SetValueCaption Method [232](#)  
 SimpleTrigger Method [232](#)  
 Step Method [235](#)  
 Stop Method (Instrument Object) [236](#)  
 Stop Method (PattgenModule Object) [236](#)  
 TestAll Method [237](#)  
 VBADisplayHelpTopic Method [237](#)  
 VBARunMacro Method [237](#)  
 VBARunRPICommand Method [238](#)  
 WaitComplete Method [238](#)  
 WriteOutput Method [242](#)

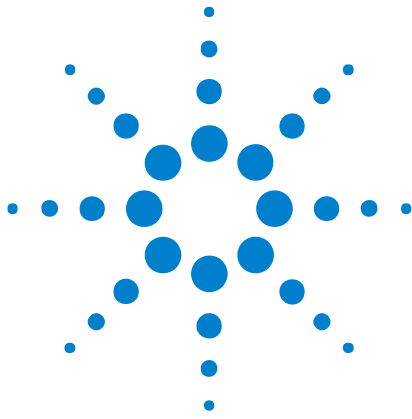
Properties	243
Activity Property	245
Axis Property	246
AxisBase Property	246
BackgroundColor Property	247
BitSize Property	248
BitSize Property (of VbaViewChartAxis)	248
Bold Property	248
BusSignalData Property	249
BusSignalType Property	249
BusSignalDifferences Property	250
BusSignals Property	250
ByteSize Property	251
Caption Property	251
CardModels Property	252
Channels Property	252
Chart Property	253
ChartType Property	253
Color Property	254
Comments Property	255
ComputerName Property	255
Count Property	256
CreatorName Property	259
Data Property	260
DataType Property	260
Description Property	260
Differences Property	261
EndSample Property	261
EndTime Property	262
FaceName Property	262
Font Property	262
Found Property	263
Frame Property	263
Frames Property	263
HasLegend Property	264
HasTitle Property	264
Instrument Property	265
IPAddress Property	265
Item Property	266
Legend Property	267
Markers Property	267

Model Property	268
Modules Property	268
Name Property	269
NumLines Property	270
OccurrencesFound Property	270
Options Property	271
Overview Property	271
PanelLocked Property	272
Polarity Property	272
Position Property	272
Position Property (of VbaViewChartLegend)	273
Probes Property	274
Reference Property	274
RemoteComputerName Property	274
RemoteUserName Property	275
RunningStatus Property	275
SampleDifferences Property	276
SampleNum Property	277
SelfTest Property	277
Setup Property	277
Size Property	278
Slot Property	278
StartSample Property	279
StartTime Property	279
Status Property	280
StatusMsg Property	281
SubrowFound Property	281
Symbols Property	281
TargetControlPort Property	282
TextColor Property	283
TimeFound Property	283
TimeFoundString Property	284
Title Property	284
Tools Property	284
Trigger Property	285
Type Property	285
Value Property	286
VBAVersion Property	287
VBE Property	287
Version Property	287
WebBrowser Property (for VbaViewWindow object)	288
WebBrowser Property (for VbaViewWebBrowser object)	288

Windows Property [289](#)  
\_NewEnum Property [289](#)

## **5 What's Changed**

### **Index**

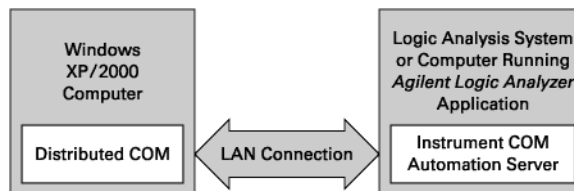


# 1

## COM Automation Overview

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++).



**Figure 1** COM Automation Architecture

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](#)).



## 2 Setting Up for COM Automation

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

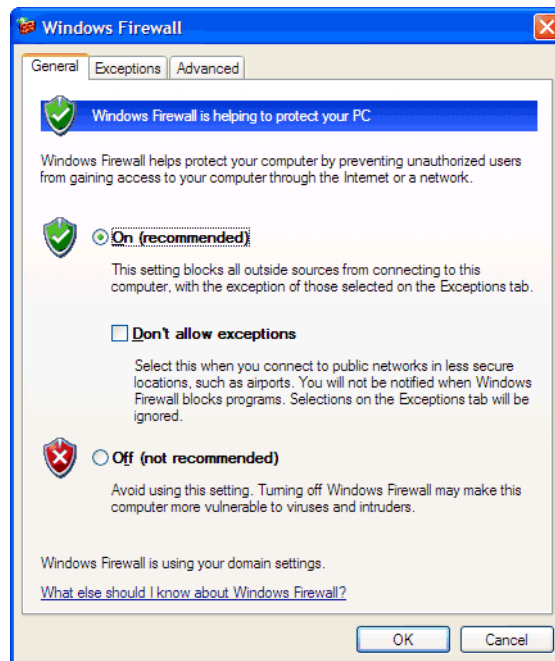
For example, you may need to change firewall settings in order to:

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

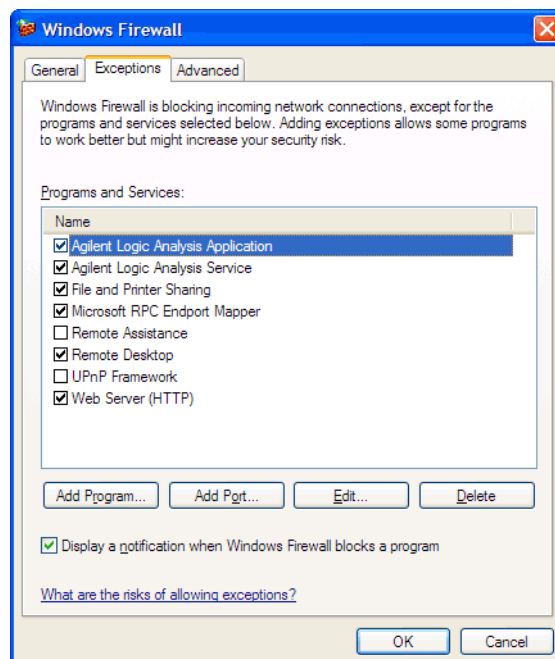
To change firewall settings to give other applications/ports access:

- 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.

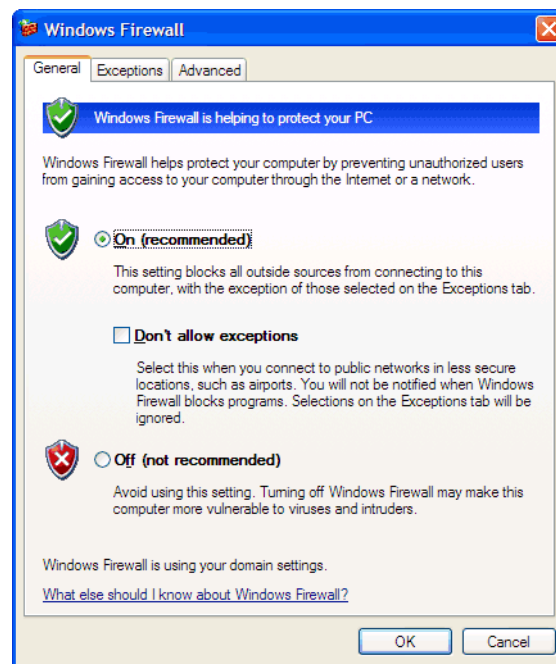
### NOTE

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.

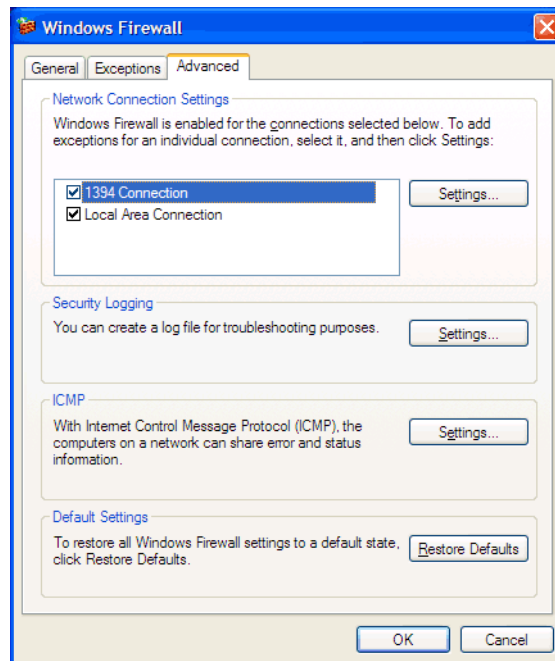
- See Also**
- To restore logic analysis system Windows Firewall defaults (see [page 18](#))

### 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.



- 4 In the Advanced tab, click **Restore Defaults** to restore the default Windows Firewall settings.
- 5 In the confirmation dialog, Click **Yes**.
- 6 Click **OK** to close the Windows Firewall dialog.
- 7 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 Endport 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

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

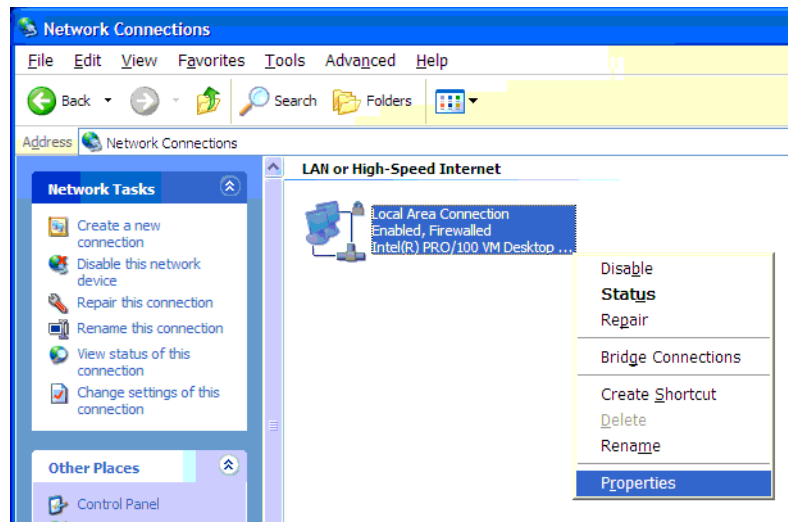
During the A.02.xx releases of the *Agilent Logic Analyzer* application, 1680-series logic analyzers and 16900-series logic analysis systems were shipped from the factory with Windows XP Professional, Service Pack 1 (SP1), and the Internet Connection Firewall was enabled and set to block accesses to 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:

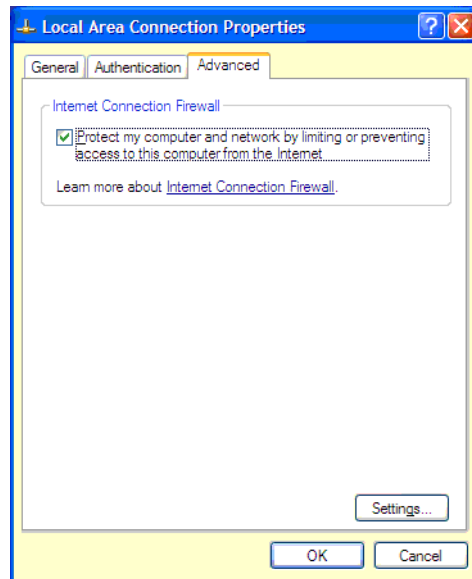
- Connect to the logic analyzer from the *Agilent Logic Analyzer* application running on another computer.
- Connect 16900A, 16901A, 16902A, or 16902B logic analysis systems together in a multiframe configuration.
- Control the logic analyzer remotely using COM automation programs.
- Use Remote Desktop 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**.

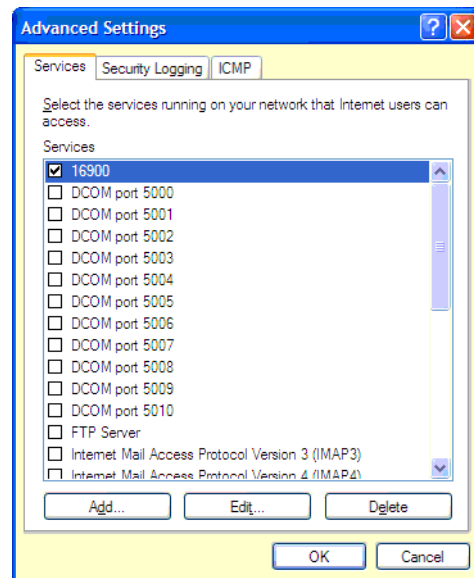


- 4 In the Advanced tab of the Local Area Connection Properties dialog, click **Settings....**



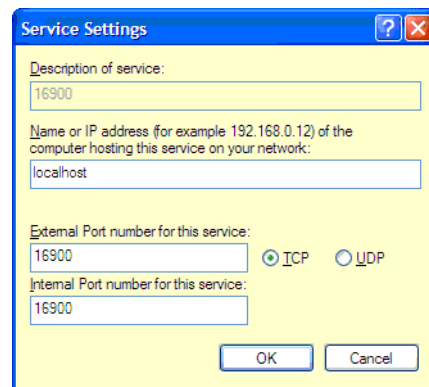
- 5 In the Services tab of the Advanced Settings dialog, check the service that you want to allow access to.

## 2 Setting Up for COM Automation



6 Click **Edit....**

7 In the Service Settings dialog, enter "localhost" in the **Name or IP address** field, and click **OK**.



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:

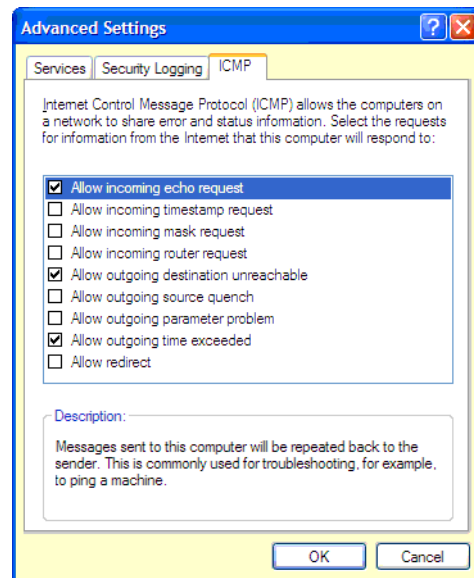
Feature	Service(s)	Notes
Remote Connect	• 16900	
Multiframe	• 16900	

COM Automation	<ul style="list-style-type: none"> <li>• 16900</li> <li>• DCOM port 5000-5010</li> <li>• Microsoft RPC endpoint mapper</li> </ul>	One DCOM port is required per DCOM connection.
Remote Desktop	<ul style="list-style-type: none"> <li>• Remote Desktop</li> </ul>	
NetOp	<ul style="list-style-type: none"> <li>• NetOp</li> </ul>	
RealVNC	<ul style="list-style-type: none"> <li>• VNC</li> </ul>	
Shared folders	<ul style="list-style-type: none"> <li>• netbios-dgm (Netbios Datagram Service -UDP)</li> <li>• netbios-ds (Direct Host over TCP/IP -TCP)</li> <li>• netbios-ds (Direct Host over TCP/IP -UDP)</li> <li>• netbios-ns (Netbios Name Service -UDP)</li> <li>• netbios-ssn (Netbios Session Service TCP)</li> <li>• netbios-ssn (Netbios Session Service UDP)</li> </ul>	
Web server	<ul style="list-style-type: none"> <li>• Web Server (HTTP)</li> </ul>	

9 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**

## 2 Setting Up for COM Automation



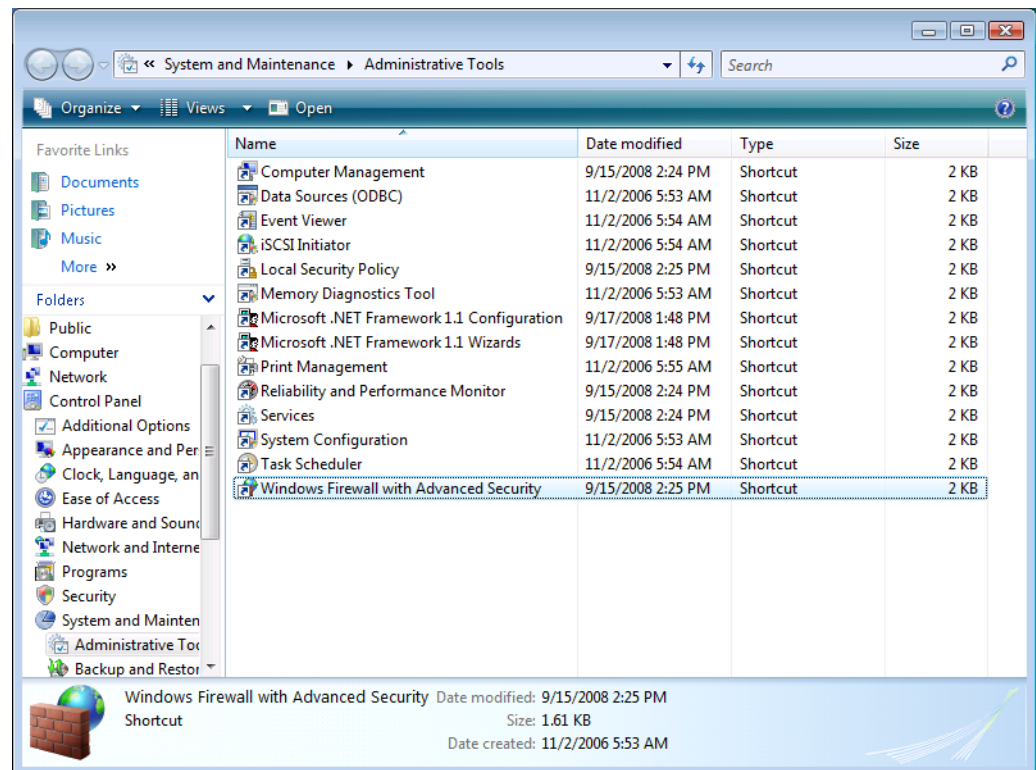
- 10 Click **OK** to close the Advanced Settings dialog.
- 11 Click **OK** to close the Local Area Connection Properties dialog.



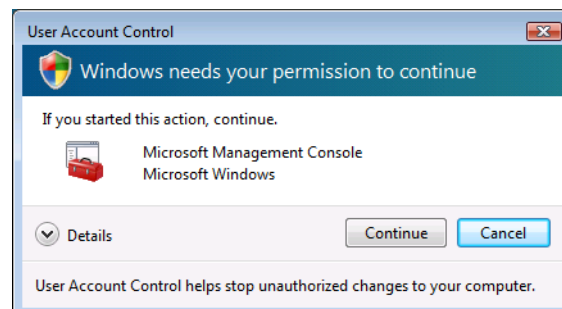
## Changing Firewall Settings, Vista Service Pack 1

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>**).

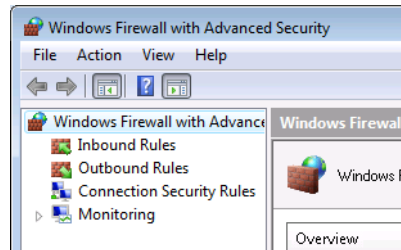


- 2 In the User Account Control dialog, click **Continue**.

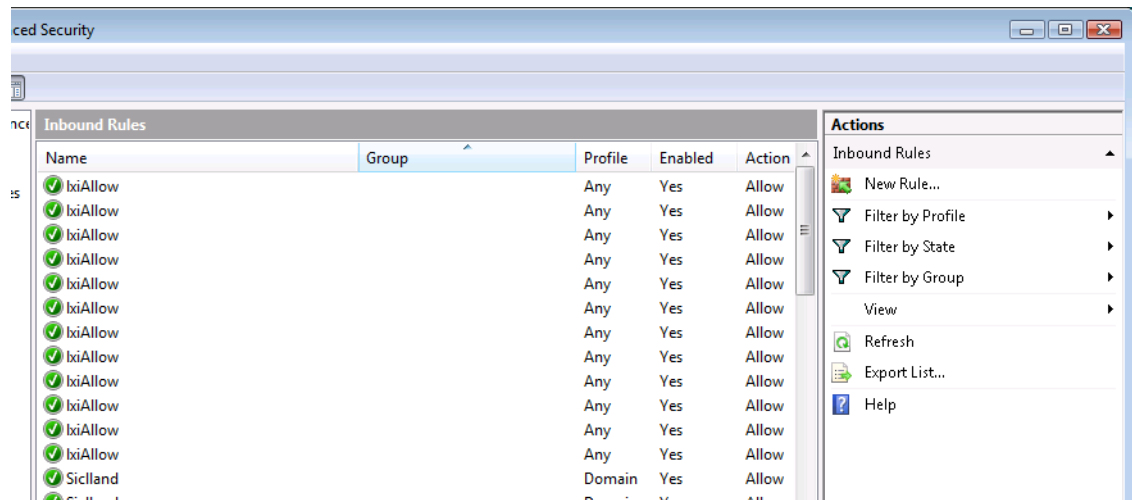


## 2 Setting Up for COM Automation

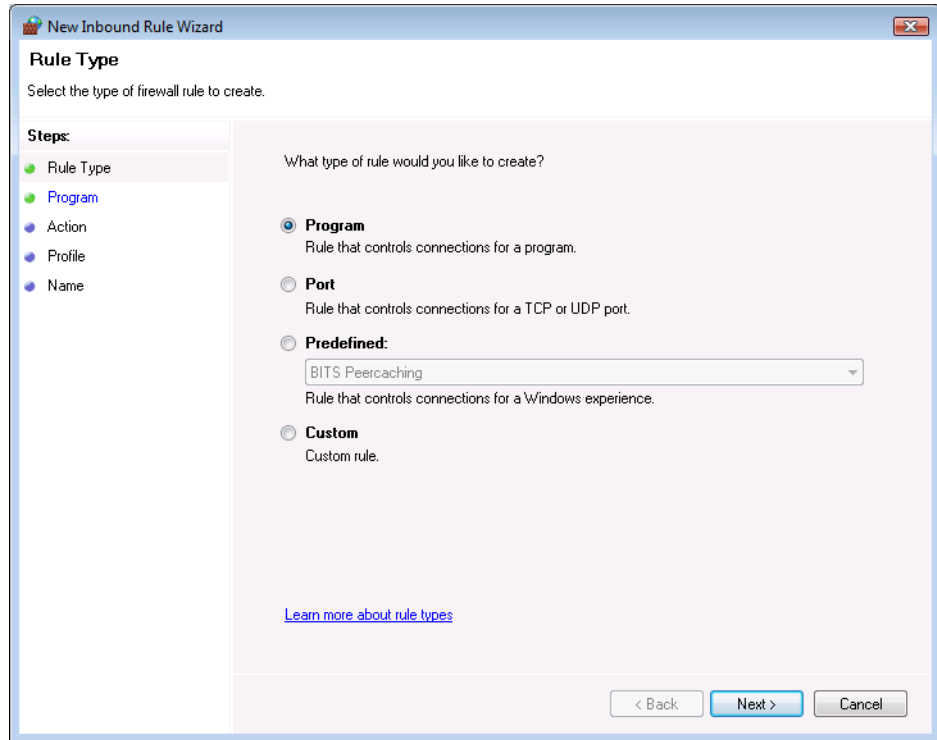
- 3 In the Windows Firewall with Advanced Security window, select **Inbound Rules**.



- 4 In the Actions pane to the right of the Inbound Rules list, select **New Rule...**

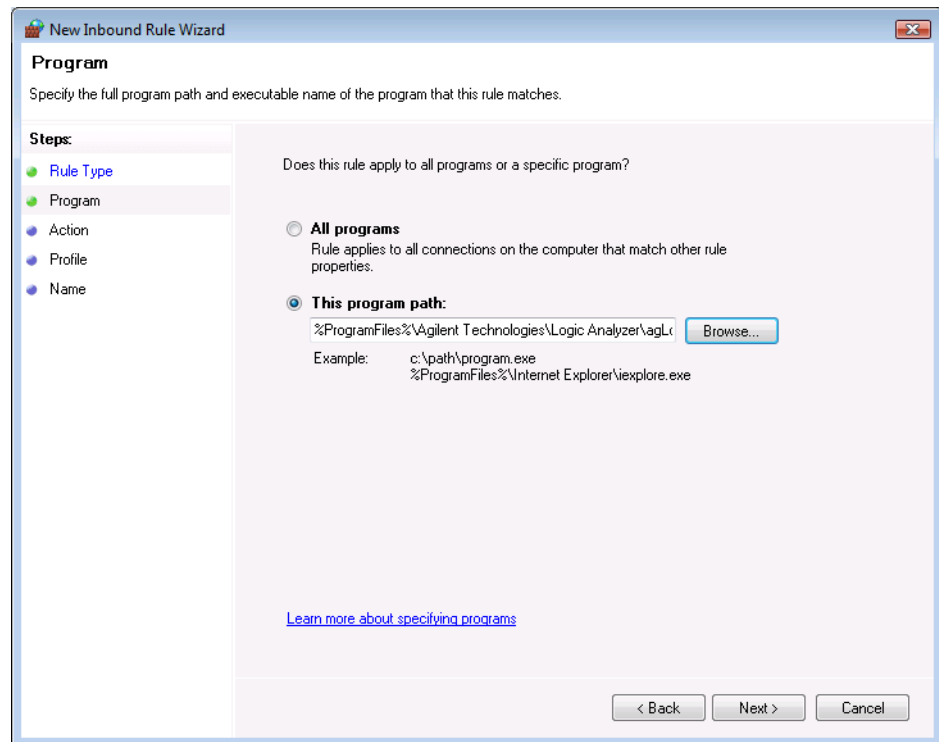


- 5 In the New Inbound Rules Wizard:
- a In the Rule Type page, select **Program**; then, click **Next >**.

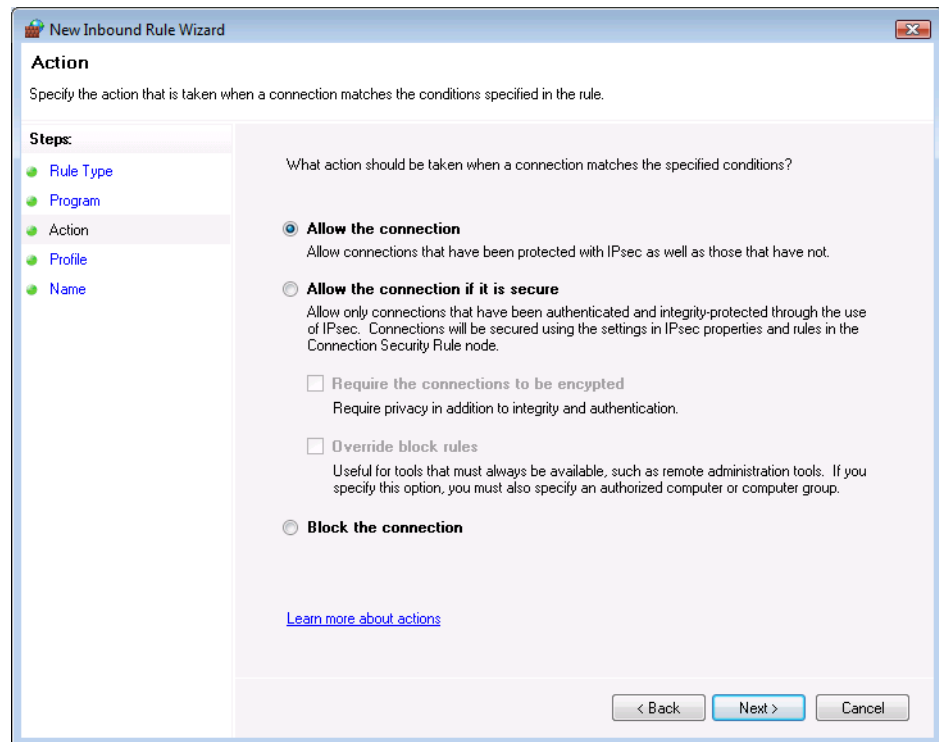


- b In the Program page, click **Browse** to select the "agLogic.exe" program (the default installation directory is C:\Program Files\Agilent Technologies\Logic Analyzer\); then, click **Next >**.

## 2 Setting Up for COM Automation

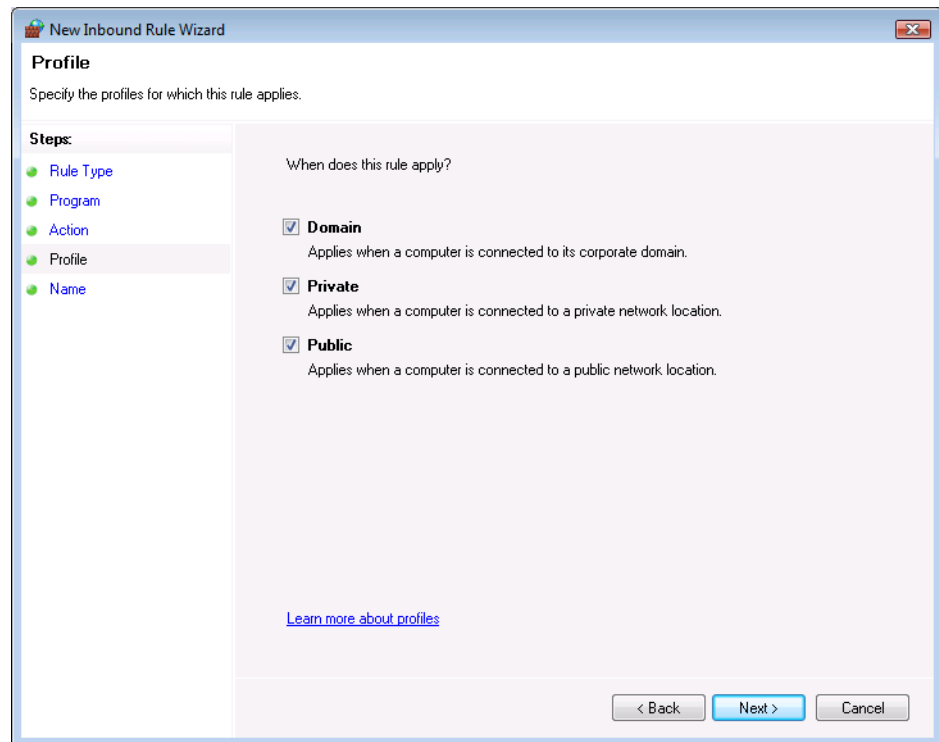


- c In the Action page, select **Allow the connection**; then, click **Next >**.

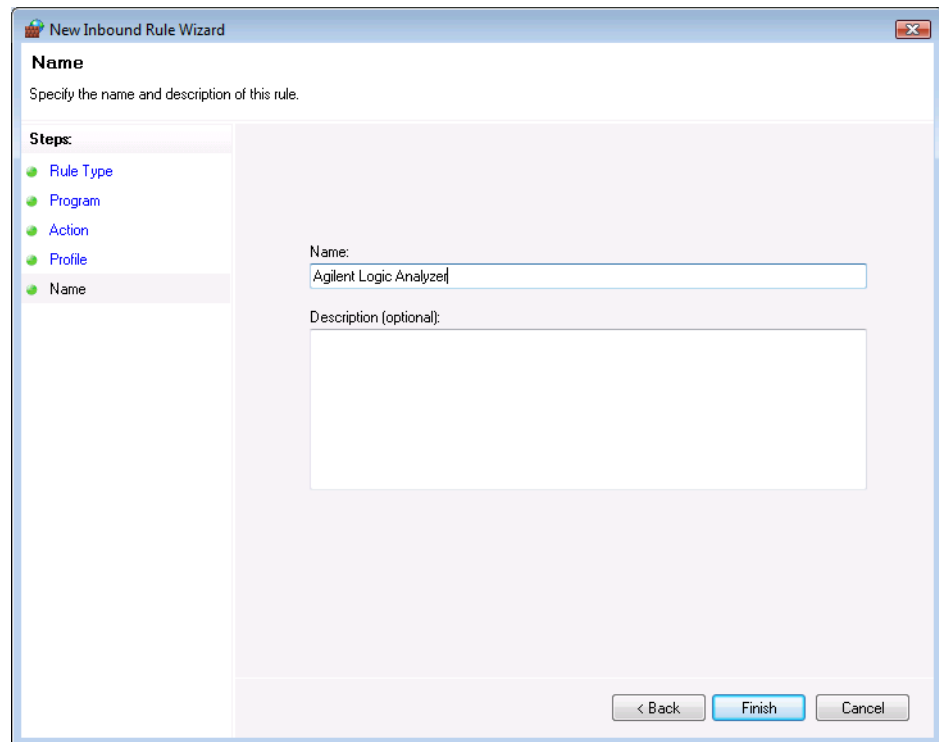


- d In the Profile page, select **Domain**, **Private**, and **Public**; then, click **Next >**.

## 2 Setting Up for COM Automation



- e 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:

Inbound Rules				
Name	Group	Profile	Enabled	Action
✓ Agilent Logic Analyzer		Any	Yes	Allow
✓ Agilent Logic Analyzer Service		Any	Yes	Allow
✓ bxiAllow		Any	Yes	Allow

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

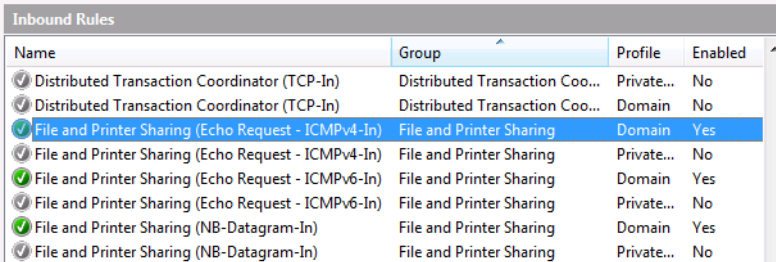
## 2 Setting Up for COM Automation

- 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.



Name	Group	Profile	Enabled
✓ Distributed Transaction Coordinator (TCP-In)	Distributed Transaction Coo...	Private...	No
✓ Distributed Transaction Coordinator (TCP-In)	Distributed Transaction Coo...	Domain	No
✓ File and Printer Sharing (Echo Request - ICMPv4-In)	File and Printer Sharing	Domain	Yes
✓ File and Printer Sharing (Echo Request - ICMPv4-In)	File and Printer Sharing	Private...	No
✓ File and Printer Sharing (Echo Request - ICMPv6-In)	File and Printer Sharing	Domain	Yes
✓ File and Printer Sharing (Echo Request - ICMPv6-In)	File and Printer Sharing	Private...	No
✓ File and Printer Sharing (NB-Datagram-In)	File and Printer Sharing	Domain	Yes
✓ File and Printer Sharing (NB-Datagram-In)	File and Printer Sharing	Private...	No

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 through 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:
  - a Check the **Enable Distributed COM on this computer** option.
  - b For the Default Authentication Level, select **Connect**.
  - c For the Default Impersonation Level, select **Identify**.
- 4 In the COM Security tab:
  - a Under Access Permissions, click **Edit Limits....**
  - b In the Access Permission dialog, make sure the Everyone account has "Allow" checked for both Local Access and Remote Access.
  - c Click **OK** to close the Access Permission dialog.
  - d Under Launch and Activation Permissions, click **Edit Limits....**
  - e 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.

- f Click **OK** to close the Launch Permission dialog.
- 5 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.

Tab	Settings
General	Authentication Level should be set to "Default".
Location	Set to "Run application on this computer".
Security	<p>Use custom launch and activation permissions. Verify "Everyone" has launch and activation permissions. If not:</p> <ul style="list-style-type: none"> <li>• Add "Everyone" and make sure "Allow" is checked for Local Launch, Remote Launch, Local Activation, and Remote Activation.</li> </ul> <p>Use custom access permissions. Verify "Everyone" has access permission. If not:</p> <ul style="list-style-type: none"> <li>• Add "Everyone" and make sure "Allow" is checked for Local Access and Remote Access.</li> </ul> <p>Use default configuration permissions.</p>
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.

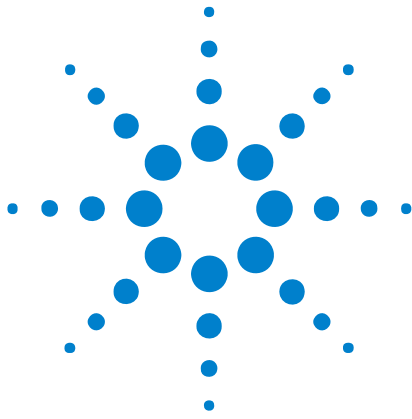
Tab	Settings
General	Authentication Level should be set to "Default".
Location	Normally, none of these options are selected, and the name of the computer on which to run the application is specified in the remote program (see the <b>Connect (see page 118)</b> object's <b>Instrument (see page 265)</b> 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.
Security	Use default launch (and activation if on Windows XP) permissions. Use default access permissions. Use Custom configuration permissions. Verify that you have "Full Control".
Endpoints	Leave at default system protocols.
Identity	Normally, this tab does not appear (but it can if the <i>Agilent Logic Analyzer</i> application has been previously installed on the remote computer). If this tab appears, set to "The interactive user".

- 4 In the Agilent 168x/169x/169xx Logic Analyzer Properties dialog, click **OK**.

## **2    Setting Up for COM Automation**

- 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

```
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
```

```

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
                ' Rows start with 1, and the bus/signal name is on the first
                ' row; so, add 2.
                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

    ' 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::IIInstrumentPtr 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;

```



```

_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' ", (char*) moduleName);
                    printf("module'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 = strlen(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 \
// 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::IIInstrumentPtr 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"
            };
        }
    }
}
```

```

        // 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;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 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 = strlen(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);
}

```

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>

////////////////////////////////////
//
// 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::IIInstrumentPtr 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);
    }
}

```

```

// 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);
}

```

**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.
//
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::IIInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Get the logic analyzer object.
            _bstr_t moduleName = "My 16910A-1";
            AgtLA::IIAnalyzerModulePtr 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>";
            pAnalyzer->PutSetup(myStateSamplingSetup);
        }
    }
}

```

```

        <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>";
pAnalyzer->PutSetup(myStateSamplingSetup);

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

```

```

        int nLen = strlen(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
DoesCurrentVersionMatchRequiredVersion = True
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
// 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 <string>
using namespace std;

/////////////////////////////////////////////////////////////////
//
// Forward declarations.
//
boolean GetNumbersForVersionString(
    _bstr_t&    strVersion,
    long&       nMajor,
    long&       nMinor,
    long&       nSubminor);

boolean DoesCurrVersionMatch(
    _bstr_t&     strReqdVersion);

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))
    {
        _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 there's no period, we need to exit.
    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);
        }
    }
}

```

```

        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\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 = strlen(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.

### NOTE

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.

```
// 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 (PrintLADData.pl).
- 3 Run the example from the Command Prompt by entering the command:  
"perl PrintLADData.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)
#   $LAAalyzer       -> 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 $LAAalyzer = $LAInst->Modules(0);

#
# Get the module's bus/signal names and store into 'LANameArray'
#
my $LABusSignals = $LAAalyzer->BusSignals();
my $LABusSignalsCount = $LABusSignals->count;
my @LANameArray;
my @LADataArrays;
my $numRows = Variant(VT_I4 | VT_BYREF, 0);
my $LStartRange = -10;
my $LEndRange = 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 $LData = $LABusSignals->Item($index)->BusSignalData;
    my $LABusSignalType = $LABusSignals->Item($index)->BusSignalType();
    my $LDataType = $LAConstants->{AgtDataLong};

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

    my $LdataArray = $LData->GetDataBySample($LStartRange,
        $LEndRange, $LDataType, $numRows);
    push(@LDataArrays, $LdataArray);
}

#
# Print the Arrays
#
PrintArrays(@LDataArrays);

```

## 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

```
#
# This Python 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 Python directly on the LA)
#   LAModule         -> 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 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:
```



```

##
## 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
### =====
### =====

```

```

###                                * 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":
#     LAAnalyzerModule = \
#         win32com.client.CastTo(LAModule, "IAnalyzerModule")
#     LAAnalyzerModule.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 'LDataArrays'.
LDataArrays = []
LDataArrays.append(LABusSignalNameArray)
LDataArrays.append(LABusSignalTypeArray)
LStartRange = -10
LEndRange = 10

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

    if LABusSignalType == constants.AgtBusSignalTime:
        LDataType = constants.AgtDateTime

    if LData.Type == "Sample":
        SampleBusSignalData = \
            win32com.client.CastTo(LData, "ISampleBusSignalData")
        (LDataArray, NumRows) = \
            SampleBusSignalData.GetDataBySample(LStartRange,
                                                LEndRange,
                                                LDataType)

        LDataArrays.append(LDataArray)

#
# Print the Arrays.
#
PrintArrays(LDataArrays)

```

## 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

```
#
# 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 ] \
           " 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 ", \
                      rwmode " $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 } \
        "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 "*****";
    }

# 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;  # DateTime
              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";

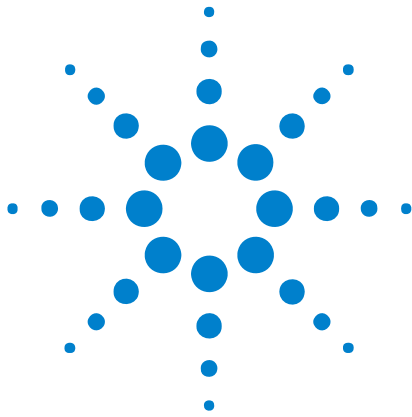
incr index;
}

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

# 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} $bus_signal_info {
        set value [$datahandle GetDataBySample $i $i $datatype numRows];
        puts -nonewline [ format "%*s " $width $value ];
    }
    puts "";
}

```





## 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

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

Objects	Methods/Properties	Description
AnalyzerModule (see <a href="#">page 106</a> )		A state/timing analyzer hardware measurement module.
methods	GetRawData (see <a href="#">page 200</a> )	Given a range, returns the raw analyzer data.
	GetRawTimingZoomData (see <a href="#">page 202</a> )	Given a range, returns the raw analyzer timing zoom data.
	RecallTriggerByName (see <a href="#">page 224</a> )	Loads a named trigger from the recall buffer.
	RecallTriggerByFile (see <a href="#">page 223</a> )	Loads a previously saved trigger file on the instrument file system.
	SimpleTrigger (see <a href="#">page 232</a> )	Trigger on a simple condition with optional occurrence and storage qualification.
	WaitComplete (see <a href="#">page 238</a> )	Waits until the analyzer module, tool, and viewer measurements are complete.
properties	Setup (see <a href="#">page 277</a> )	Gets or sets the logic analyzer's XML-format setup specification.
	Trigger (see <a href="#">page 285</a> )	Gets or sets the logic analyzer's XML-format trigger specification.
BusSignal (see <a href="#">page 108</a> )		A named and grouped set of pod channels.
methods	IsTimingZoom (see <a href="#">page 215</a> )	Is this a timing zoom bus/signal?

Objects	Methods/Properties	Description
properties	Activity (see <a href="#">page 245</a> )	Gets the activity indicators of the bus/signal.
	BitSize (see <a href="#">page 248</a> )	Gets the number of channels in the bus/signal.
	BusSignalData (see <a href="#">page 249</a> )	Gets the acquisition data associated with a bus/signal.
	BusSignalType (see <a href="#">page 249</a> )	Gets the type of bus/signal.
	ByteSize (see <a href="#">page 251</a> )	Gets the size of the bus/signal in bytes.
	Channels (see <a href="#">page 252</a> )	Gets the channels defined in the bus/signal.
	CreatorName (see <a href="#">page 259</a> )	Gets the name of the module, tool, or viewer that created this bus/signal.
	Name (see <a href="#">page 269</a> )	Gets or sets the name of the bus/signal.
	Polarity (see <a href="#">page 272</a> )	Gets the polarity of the bus/signal.
	Symbols (see <a href="#">page 281</a> )	Gets or sets the symbols associated with a bus/signal.
BusSignalData (see <a href="#">page 108</a> )		A generic bus/signal data object.
properties	Type (see <a href="#">page 285</a> )	Gets the specific bus/signal data type.
BusSignalDifference (see <a href="#">page 109</a> )		Represents the different values for a particular bus/signal within a sample that has differences.
properties	Name (see <a href="#">page 269</a> )	Gets the bus/signal name associated with the sample difference.
	Reference (see <a href="#">page 274</a> )	Gets the reference buffer value associated with the sample difference.
	Value (see <a href="#">page 286</a> )	Gets the data value associated with the sample difference.
BusSignalDifferences (see <a href="#">page 109</a> )		A collection object that contains all of the SampleDifference object's buses/signals with differences.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of bus/signal differences in the collection.
	Item (see <a href="#">page 266</a> )	Given an index into the collection, gets a BusSignalDifference (see <a href="#">page 109</a> ) object from the collection.
BusSignals (see <a href="#">page 113</a> )		A collection of the hardware module's defined BusSignals.

Objects	Methods/Properties	Description
methods	Add (see <a href="#">page 168</a> )	Adds a new bus/signal to the collection.
	Remove (see <a href="#">page 224</a> )	Removes a bus/signal from the collection.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of BusSignal (see <a href="#">page 108</a> ) objects in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the BusSignal (see <a href="#">page 108</a> ) objects in the collection given either an index or name.
CompareWindow (see <a href="#">page 118</a> )		A window that compares bus/signal data.
methods	Execute (see <a href="#">page 178</a> )	Executes the compare using the current options.
properties	Options (see <a href="#">page 271</a> )	Gets or sets the Compare window options.
	SampleDifferences (see <a href="#">page 276</a> )	Gets a collection of all the samples with differences found in the last comparison.
Connect (see <a href="#">page 118</a> )		A connection to the logic analyzer instrument.
methods	CopyFile (see <a href="#">page 173</a> )	Copies a file to the instrument file system.
	GetRemoteInfo (see <a href="#">page 203</a> )	Gets the logic analyzer's remote user login and computer name.
properties	Instrument (see <a href="#">page 265</a> )	Gets the logic analyzer instrument object.
ConnectSystem (see <a href="#">page 119</a> )		A connection to the logic analyzer system.
methods	Connect (see <a href="#">page 172</a> )	Connects to the remote logic analyzer system.
	RecvFile (see <a href="#">page 224</a> )	Copies a file from the remote logic analyzer system to your local system.
	SendFile (see <a href="#">page 229</a> )	Copies a file from your local system to the remote logic analyzer system.
FindResult (see <a href="#">page 119</a> )		Gets the results from the Find (see <a href="#">page 184</a> ), FindNext (see <a href="#">page 188</a> ), and FindPrev (see <a href="#">page 189</a> ) method calls.
properties	Found (see <a href="#">page 263</a> )	Gets the found status.
	OccurrencesFound (see <a href="#">page 270</a> )	Gets the number of occurrences found.
	SubrowFound (see <a href="#">page 281</a> )	Gets the subrow number if found on a subrow.
	TimeFound (see <a href="#">page 283</a> )	Gets the time found as a double.
	TimeFoundString (see <a href="#">page 284</a> )	Gets the time found as a string.
Frame (see <a href="#">page 120</a> )		A logic analyzer frame.

Objects	Methods/Properties	Description
properties	ComputerName (see <a href="#">page 255</a> )	Gets the computer name of the logic analyzer frame.
	Description (see <a href="#">page 260</a> )	Gets a description of the logic analyzer frame.
	IPAddress (see <a href="#">page 265</a> )	Gets the frame's IP address(es).
	TargetControlPort (see <a href="#">page 282</a> )	Gets or sets the target control port value.
Frames (see <a href="#">page 120</a> )		A collection of logic analyzer frames connected via the multiframe connector.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of frames in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the frames in the collection given either an index, computer name, or IP address.
Instrument (see <a href="#">page 121</a> )		The logic analyzer instrument.
methods	Close (see <a href="#">page 172</a> )	Closes the current configuration.
	DeleteFile (see <a href="#">page 173</a> )	Deletes a file on the instrument file system.
	DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
	Export (see <a href="#">page 179</a> )	Exports data to a file on the instrument file system.
	ExportEx (see <a href="#">page 180</a> )	Exports data to a file on the instrument file system.
	GetProbeByName (see <a href="#">page 200</a> )	Given a probe name, returns its corresponding probe object.
	GetModuleByName (see <a href="#">page 196</a> )	Given a module name, returns its corresponding hardware module object.
	GetToolByName (see <a href="#">page 205</a> )	Given a tool name, returns its corresponding tool object.
	GetWindowByName (see <a href="#">page 206</a> )	Given a window name, returns its corresponding window object.
	Import (see <a href="#">page 212</a> )	Imports data from a file located on the instrument file system.
	ImportEx (see <a href="#">page 213</a> )	Imports data from a file located on the instrument file system into a particular module.
	GoOffline (see <a href="#">page 207</a> )	Disconnects the user interface from the logic analyzer frame.

Objects	Methods/Properties	Description
methods (cont'd)	GoOnline (see <a href="#">page 211</a> )	Connects the user interface to a specific logic analyzer frame.
	IsOnline (see <a href="#">page 214</a> )	Tells whether the user interface is connected to a logic analyzer frame.
	New (see <a href="#">page 215</a> )	Creates a new instrument Overview.
	Open (see <a href="#">page 216</a> )	Loads a previously saved configuration file on the instrument file system.
	PanelLock (see <a href="#">page 216</a> )	Disables user access to the instrument front panel or remote display.
	PanelUnlock (see <a href="#">page 220</a> )	Re-enables user access to the instrument front panel or remote display.
	QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.
	Run (see <a href="#">page 227</a> )	Starts running all modules.
	Save (see <a href="#">page 228</a> )	Saves the current configuration to a file on the instrument file system.
	Stop (see <a href="#">page 236</a> )	Stops all currently running modules.
	VBADisplayHelpTopic (see <a href="#">page 237</a> )	Displays the help page and topic for an installed VBA project.
	VBARunMacro (see <a href="#">page 237</a> )	Runs the specified VBA macro as if that macro was selected in the Macros dialog box.
	VBARunRPICommand (see <a href="#">page 238</a> )	Runs an ASCII RPI command in VBA.
	WaitComplete (see <a href="#">page 238</a> )	Waits until all module, tool, and viewer measurements are complete.

Objects	Methods/Properties	Description
properties	Frames (see <a href="#">page 263</a> )	Gets a collection of logic analyzer frames connected via the multiframe connector.
	Markers (see <a href="#">page 267</a> )	Gets a collection of all the markers in the instrument.
	Model (see <a href="#">page 268</a> )	Gets the model number.
	Modules (see <a href="#">page 268</a> )	Gets a collection of all the hardware modules in the instrument.
	Overview (see <a href="#">page 271</a> )	Gets the XML-format Overview window specification.
	PanelLocked (see <a href="#">page 272</a> )	Indicates the front panel is locked. If locked, the message is returned.
	Probes (see <a href="#">page 274</a> )	Gets a collection of all currently defined probes.
	RemoteComputerName (see <a href="#">page 274</a> )	Gets or sets the remote computer name.
	RemoteUserName (see <a href="#">page 275</a> )	Gets or sets the remote user login name.
	SelfTest (see <a href="#">page 277</a> )	Gets the SelfTest object.
	Status (see <a href="#">page 280</a> )	Gets the status of all hardware modules.
	Tools (see <a href="#">page 284</a> )	Gets a collection of all active software tools.
	VBAVersion (see <a href="#">page 287</a> )	Gets the version number of VBA.
	VBE (see <a href="#">page 287</a> )	Gets the VBE extensibility object.
	Version (see <a href="#">page 287</a> )	Gets the version number of the system software.
	Windows (see <a href="#">page 289</a> )	Gets a collection of all active windows.
Marker (see <a href="#">page 123</a> )		A reference point in the captured data.
properties	Comments (see <a href="#">page 255</a> )	Gets or sets the marker comments.
	Name (see <a href="#">page 269</a> )	Gets or sets the name of the marker.
	TextColor (see <a href="#">page 283</a> )	Gets or sets the marker text color.
	BackgroundColor (see <a href="#">page 247</a> )	Gets or sets the marker background color.
	Position (see <a href="#">page 272</a> )	Gets or sets the marker position.
Markers (see <a href="#">page 123</a> )		A collection of all the defined markers.

Objects	Methods/Properties	Description
methods	Add (see <a href="#">page 169</a> )	Adds a new marker to the collection using specific values.
	AddXML (see <a href="#">page 170</a> )	Adds multiple markers to the collection using an XML string.
	Remove (see <a href="#">page 225</a> )	Removes a marker from the collection.
	RemoveAll (see <a href="#">page 225</a> )	Removes all markers from the collection.
	RemoveXML (see <a href="#">page 225</a> )	Removes multiple markers from the collection using an XML string.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of markers in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the markers in the collection given either an index or name.
Module (see <a href="#">page 128</a> )		A generic hardware module.
methods	DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.
	WaitComplete (see <a href="#">page 238</a> )	Waits until the module, tool, and viewer measurements are complete.
properties	BusSignals (see <a href="#">page 250</a> )	Gets a collection of the module's defined bus/signals.
	CardModels (see <a href="#">page 252</a> )	Gets the card model numbers.
	Description (see <a href="#">page 260</a> )	Gets a description of the module.
	Frame (see <a href="#">page 263</a> )	Gets the frame in which the module resides.
	Model (see <a href="#">page 268</a> )	Gets the model number.
	Name (see <a href="#">page 269</a> )	Gets or sets the name of the module.
	RunningStatus (see <a href="#">page 275</a> )	Gets the detailed running status of the module.
	Slot (see <a href="#">page 278</a> )	Gets the module's slot location in the frame.
	Status (see <a href="#">page 280</a> )	Gets the status of the module.
	StatusMsg (see <a href="#">page 281</a> )	Gets the formatted status message.
	Type (see <a href="#">page 285</a> )	Gets the specific module type.



Objects	Methods/Properties	Description
Modules (see <a href="#">page 129</a> )		A collection of all the hardware modules installed in the instrument.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of modules in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the modules in a collection given either a slot, index, or name.
PattgenModule (see <a href="#">page 129</a> )		A pattern generator hardware module.
methods	GetLine (see <a href="#">page 194</a> )	Gets an instruction or vector at line number.
	GetLineLabel (see <a href="#">page 196</a> )	Gets a vector's label value at line number.
	InsertLine (see <a href="#">page 214</a> )	Inserts a new instruction or vector after line number.
	RemoveLine (see <a href="#">page 226</a> )	Removes the instruction or vector at line number.
	Reset (see <a href="#">page 226</a> )	Resets the current line number to the first line.
	Resume (see <a href="#">page 227</a> )	Resumes running the pattern generator from the current line number.
	Run (see <a href="#">page 228</a> )	Starts running the pattern generator.
	SetLine (see <a href="#">page 230</a> )	Sets an instruction or vector at line number.
	SetLineLabel (see <a href="#">page 230</a> )	Sets a vector's label value at line number.
	Step (see <a href="#">page 235</a> )	Steps the pattern generator from the current line number.
	Stop (see <a href="#">page 236</a> )	Stops the pattern generator if it is currently running.
properties	NumLines (see <a href="#">page 270</a> )	Gets the number of lines in the main sequence.
Probe (see <a href="#">page 135</a> )		A generic probe.
methods	DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.
properties	Name (see <a href="#">page 269</a> )	Gets or sets the name of the probe.
	Type (see <a href="#">page 285</a> )	Gets the probe's type.
Probes (see <a href="#">page 135</a> )		A collection of all active probes.

Objects	Methods/Properties	Description
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of probes in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the probes in the collection given either an index or name.
SampleBusSignalData (see <a href="#">page 138</a> )		The data for a specific bus/signal captured in the state or timing sampling modes.
methods	GetDataBySample (see <a href="#">page 189</a> )	Given a sample range, returns an array of data.
	GetDataByTime (see <a href="#">page 192</a> )	Given a time range, returns an array of data.
	GetNumSamples (see <a href="#">page 199</a> )	Given a range, returns the number of samples stored.
	GetSampleNumByTime (see <a href="#">page 204</a> )	Gets the closest sample number corresponding to the time given.
	GetTime (see <a href="#">page 204</a> )	Given a range, returns the time for this bus/signal in the format specified by the data type given.
properties	DataType (see <a href="#">page 260</a> )	Gets the recommended bus/signal data type.
	EndSample (see <a href="#">page 261</a> )	Gets the data's ending sample number relative to trigger.
	EndTime (see <a href="#">page 262</a> )	Gets the data's ending time relative to trigger.
	StartSample (see <a href="#">page 279</a> )	Gets the data's starting sample number relative to trigger.
	StartTime (see <a href="#">page 279</a> )	Gets the data's starting time relative to trigger.
SampleDifference (see <a href="#">page 149</a> )		Represents a sample containing a CompareWindow difference.
properties	BusSignalDifferences (see <a href="#">page 250</a> )	Gets a collection of all the buses/signals with differences for this sample.
	SampleNum (see <a href="#">page 277</a> )	Gets the sample number at which differences occurred.
SampleDifferences (see <a href="#">page 149</a> )		A collection object of all the sample differences in the CompareWindow object.

Objects	Methods/Properties	Description
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of bus/signal differences in the collection.
	Item (see <a href="#">page 266</a> )	Given an index into the collection, gets a SampleDifference (see <a href="#">page 149</a> ) object from the collection.
SelfTest (see <a href="#">page 149</a> )		Object for running instrument self-tests.
methods	TestAll (see <a href="#">page 237</a> )	Runs an instrument's self-tests.
Tool (see <a href="#">page 152</a> )		A generic instrument software tool.
methods	DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.
properties	BusSignals (see <a href="#">page 250</a> )	Gets a collection of the tool's defined bus/signals.
	Name (see <a href="#">page 269</a> )	Gets or sets the name of the tool.
	Type (see <a href="#">page 285</a> )	Gets the specific tool type.
Tools (see <a href="#">page 153</a> )		A collection of all of the instrument's software tools.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of tools in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the tools in the collection given either an index or name.
VbaViewChart (see <a href="#">page 156</a> )		A chart for the VbaViewWindow.
methods	Draw (see <a href="#">page 178</a> )	Draws the chart.
properties	Axis (see <a href="#">page 246</a> )	Gets the chart axis given an axis type.
	ChartType (see <a href="#">page 253</a> )	Gets or sets the chart type.
	Data (see <a href="#">page 260</a> )	Gets the chart data.
	HasLegend (see <a href="#">page 264</a> )	Gets or sets if the legend is visible.
	HasTitle (see <a href="#">page 264</a> )	Gets or sets if the title is visible.
	Legend (see <a href="#">page 267</a> )	Gets the chart legend.
	Title (see <a href="#">page 284</a> )	Gets the title of the chart.
VbaViewChartAxis (see <a href="#">page 157</a> )		The axis of a VbaViewChart.

Objects	Methods/Properties	Description
properties	AxisBase (see <a href="#">page 246</a> )	Gets or sets the chart axis base.
	BitSize (see <a href="#">page 248</a> )	Gets or sets the width of the data in bits. This is used to format the Axis values.
	HasTitle (see <a href="#">page 264</a> )	Gets or sets if the title is visible.
	Title (see <a href="#">page 284</a> )	Gets the title of the axis.
VbaViewChartData (see <a href="#">page 157</a> )		The data of a VbaViewChart.
methods	AddPointArrays (see <a href="#">page 170</a> )	Adds an array of points to the chart. This is only valid for chart types AgtChartTypeLine and AgtChartTypeXYScatter.
	Clear (see <a href="#">page 171</a> )	Clears all of the chart data.
	GetGroupCaption (see <a href="#">page 193</a> )	Gets the caption associated with a group (row).
	GetValueCaption (see <a href="#">page 206</a> )	Gets the caption associated with all values at index (column).
	SetGroupCaption (see <a href="#">page 229</a> )	Sets the caption associated with a group (row).
	SetValue (see <a href="#">page 231</a> )	Sets an individual value in the chart array.
	SetValueArray (see <a href="#">page 231</a> )	Sets an array of values in the chart array starting at index 0.
	SetValueCaption (see <a href="#">page 232</a> )	Sets the caption associated with all values at index (column).
VbaViewChartFont (see <a href="#">page 158</a> )		The font of a VbaViewChartTitle.
properties	Bold (see <a href="#">page 248</a> )	Gets or sets the text thickness.
	Color (see <a href="#">page 254</a> )	Gets or sets the text color.
	FaceName (see <a href="#">page 262</a> )	Gets or sets the text face name string.
	Size (see <a href="#">page 278</a> )	Gets or sets the text size.
VbaViewChartLegend (see <a href="#">page 158</a> )		The legend of a VbaViewChart.
properties	Position (see <a href="#">page 273</a> )	Gets or sets the chart legend position.
VbaViewChartTitle (see <a href="#">page 159</a> )		The title of a VbaViewChart.
properties	Caption (see <a href="#">page 251</a> )	Gets or sets the chart title caption.
	Font (see <a href="#">page 262</a> )	Gets the chart title font.
VbaViewWebBrowser (see <a href="#">page 159</a> )		A web browser for the VbaViewWindow.
methods	Clear (see <a href="#">page 171</a> )	Displays an empty web page.
properties	WebBrowser (see <a href="#">page 288</a> )	Gets the contained IWebBrowser2 interface.

Objects	Methods/Properties	Description
VbaViewWindow (see <a href="#">page 160</a> )		A window for Visual Basic for Applications (VBA) program output.
methods	ClearOutput (see <a href="#">page 172</a> )	Clears the strings from the output window.
	WriteOutput (see <a href="#">page 242</a> )	Writes a string to the output window.
properties	Chart (see <a href="#">page 253</a> )	Gets the Chart view.
	WebBrowser (see <a href="#">page 288</a> )	Gets the Web Browser view.
Window (see <a href="#">page 160</a> )		A generic instrument display window.
methods	DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
	Find (see <a href="#">page 184</a> )	Finds a specified data event with optional occurrence and time duration.
	FindNext (see <a href="#">page 188</a> )	Finds the next event by searching forward from the last event found using the event specified by the last call to Find.
	FindPrev (see <a href="#">page 189</a> )	Finds the previous event by searching backward from the last event found using the event specified by the last call to Find.
	GoToPosition (see <a href="#">page 212</a> )	Moves the center of the window to a new position.
	QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.
properties	BusSignals (see <a href="#">page 250</a> )	Gets a collection of the window's defined bus/signals.
	Name (see <a href="#">page 269</a> )	Gets or sets the name of the window.
	Type (see <a href="#">page 285</a> )	Gets the window's type.
Windows (see <a href="#">page 161</a> )		A collection of all of the instrument's display windows.
properties	_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 256</a> )	Gets the number of windows in the collection.
	Item (see <a href="#">page 266</a> )	Gets one of the windows in the collection given either an index or name.

**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](#))

*Italics = This denotes a generic/specific relationship known as inheritance.*

### Containment and Inheritance

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:

```
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

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

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

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

**See Also** • Logic Analyzer Object Hierarchy Overview (see [page 102](#))

## AnalyzerModule Object

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

**To Access** •

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

**Description** The **AnalyzerModule** object represents a state/timing analyzer hardware measurement 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 AnalyzerModule specific object.
Dim myAnalyzer As AgtLA.AnalyzerModule
Set myAnalyzer = myInst.Modules(0)
MsgBox "Trigger: " + myAnalyzer.Trigger
```

## Methods

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

(Also Includes Module (see [page 128](#)) object methods)

## Properties

Property	Description
Setup (see <a href="#">page 277</a> )	Gets or sets the logic analyzer's XML-format setup specification.
Trigger (see <a href="#">page 285</a> )	Gets or sets the logic analyzer's XML-format trigger specification.

(Also Includes Module (see [page 128](#)) object properties)

## BusSignal Object

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

- To Access**
- BusSignals.Item (see [page 266](#)) IndexOrName
  - BusSignals IndexOrName

### Methods

Method	Description
IsTimingZoom (see <a href="#">page 215</a> )	Is this a timing zoom bus/signal?

### Properties

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

- See Also**
- BusSignals (see [page 113](#)) object

## BusSignalData Object

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

- To Access**
- BusSignal.BusSignalData (see [page 249](#))

**Methods** There are no methods.

### Properties

Property	Description
Type (see <a href="#">page 285</a> )	Gets the specific bus/signal data type.

**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

Property	Description
Name (see <a href="#">page 269</a> )	Gets the bus/signal name associated with the sample difference.
Reference (see <a href="#">page 274</a> )	Gets the reference buffer value associated with the sample difference.
Value (see <a href="#">page 286</a> )	Gets the data value associated with the sample difference.

**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

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

**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:
//

```

```

//      - 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::IIstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Load the configuration file.
            _bstr_t configFile = "C:\\LA\\Configs\\compare.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 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;

```



```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 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 = strlen(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

Method	Description
Add (see <a href="#">page 168</a> )	Adds a new bus/signal to the collection.
Remove (see <a href="#">page 224</a> )	Removes a bus/signal from the collection.

**Properties**

Property	Description
_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 256</a> )	Gets the number of BusSignal (see <a href="#">page 108</a> ) objects in the collection.
Item (see <a href="#">page 266</a> )	Gets one of the BusSignal (see <a href="#">page 108</a> ) objects in the collection given either an index or name.

**Remarks****NOTE**

The "Time" data is no longer returned as a bus/signal (see What's Changed (see [page 293](#))).

**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)
    myString = myString + ", ByteSize=" + Str(myBusSignal.ByteSize)
    myString = myString + ", Type="
    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 myBusSignal
```

```

        myString = myString + " Channels=" + _
            myBusSignal.Channels + vbNewLine
        myString = myString + " Polarity=" + _
            myBusSignal.Polarity + vbNewLine
        myString = myString + " Activity=" + _
            myBusSignal.Activity + vbNewLine
    Case Else
        myString = myString + "Unknown" + vbNewLine
    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 \
// 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::IIstrumentPtr 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 =
                pModule->GetBusSignals()->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:
                {

```

```

        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 = strlen(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

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

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

### Methods

Method	Description
Execute (see <a href="#">page 178</a> )	Executes the compare using the current options.

(Also Includes Window (see [page 160](#)) methods)

### Properties

Property	Description
Options (see <a href="#">page 271</a> )	Gets or sets the Compare window options.
SampleDifferences (see <a href="#">page 276</a> )	Gets a collection of all the samples with differences found in the last comparison.

(Also Includes Window (see [page 160](#)) properties)

## Connect Object

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

### To Access

```
Dim variable As AgtLA.Connect
Set variable = CreateObject("AgtLA.Connect")
```

### Methods

Method	Description
CopyFile (see <a href="#">page 173</a> )	Copies a file to the instrument file system.
GetRemoteInfo (see <a href="#">page 203</a> )	Gets the logic analyzer's remote user login and computer name.

**Properties**

Property	Description
Instrument (see <a href="#">page 265</a> )	Gets the logic analyzer instrument object.

**ConnectSystem Object**

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

**To Access**

•

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

**Methods**

Method	Description
Connect (see <a href="#">page 172</a> )	Connects to the remote logic analyzer system.
RecvFile (see <a href="#">page 224</a> )	Copies a file from the remote logic analyzer system to your local system.
SendFile (see <a href="#">page 229</a> )	Copies a file from your local system to the remote logic analyzer system.

**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**

Property	Description
Found (see <a href="#">page 263</a> )	Gets the found status.
OccurrencesFound (see <a href="#">page 270</a> )	Gets the number of occurrences found.
SubrowFound (see <a href="#">page 281</a> )	Gets the subrow number if found on a subrow.
TimeFound (see <a href="#">page 283</a> )	Gets the time found as a double.
TimeFoundString (see <a href="#">page 284</a> )	Gets the time found as a string.

- See Also**
- Find (see [page 184](#)) method
  - FindNext (see [page 188](#)) method
  - FindPrev (see [page 189](#)) method
  - GetDataByTime (see [page 192](#)) method

## Frame Object

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

- To Access**
- Frames.Item (see [page 266](#)) IndexOrName
  - Frames IndexOrName

**Methods** There are no methods.

### Properties

Property	Description
ComputerName (see <a href="#">page 255</a> )	Gets the computer name of the logic analyzer frame.
Description (see <a href="#">page 260</a> )	Gets a description of the logic analyzer frame.
IPAddress (see <a href="#">page 265</a> )	Gets the frame's IP address(es).
TargetControlPort (see <a href="#">page 282</a> )	Gets or sets the target control port value.

- See Also**
- Frames (see [page 120](#)) object

## Frames Object

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

- To Access**
- Instrument.Frames (see [page 263](#))

**Methods** There are no methods.

### Properties

Property	Description
_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 256</a> )	Gets the number of frames in the collection.
Item (see <a href="#">page 266</a> )	Gets one of the frames in the collection given either an index, computer name, or IP address.

- 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

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

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

## Properties

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

Property	Description
Status (see <a href="#">page 280</a> )	Gets the status of all hardware modules.
Tools (see <a href="#">page 284</a> )	Gets a collection of all active software tools.
VBAVersion (see <a href="#">page 287</a> )	Gets the version number of VBA.
VBE (see <a href="#">page 287</a> )	Gets the VBE extensibility object.
Version (see <a href="#">page 287</a> )	Gets the version number of the system software.
Windows (see <a href="#">page 289</a> )	Gets a collection of all active windows.

**See Also** • Instrument (see [page 265](#)) property

## Marker Object

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

**To Access** • Markers.Item (see [page 266](#)) IndexOrName  
• Markers IndexOrName

**Methods** There are no methods.

### Properties

Property	Description
Comments (see <a href="#">page 255</a> )	Gets or sets the marker comments.
Name (see <a href="#">page 269</a> )	Gets or sets the name of the marker.
TextColor (see <a href="#">page 283</a> )	Gets or sets the marker text color.
BackgroundColor (see <a href="#">page 247</a> )	Gets or sets the marker background color.
Position (see <a href="#">page 272</a> )	Gets or sets the marker position.

**See Also** • Markers (see [page 123](#)) object

## Markers Object

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

**To Access** • Instrument.Markers (see [page 267](#))

**Methods**

Method	Description
Add (see <a href="#">page 169</a> )	Adds a new marker to the collection using specific values.
AddXML (see <a href="#">page 170</a> )	Adds multiple markers to the collection using an XML string.
Remove (see <a href="#">page 225</a> )	Removes a marker from the collection.
RemoveAll (see <a href="#">page 225</a> )	Removes all markers from the collection.
RemoveXML (see <a href="#">page 225</a> )	Removes multiple markers from the collection using an XML string.

**Properties**

Property	Description
_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 256</a> )	Gets the number of markers in the collection.
Item (see <a href="#">page 266</a> )	Gets one of the markers in the collection given either an index or name.

**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 \
// 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::IIstrumentPtr 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' /> \
            ";
        }
    }
}

```

```

        <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("*** 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 = strlen(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

Method	Description
<code>DoAction</code> (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
<code>DoCommands</code> (see <a href="#">page 174</a> )	Execute a particular XML-based command.
<code>QueryCommand</code> (see <a href="#">page 220</a> )	Query for XML-based commands.
<code>WaitComplete</code> (see <a href="#">page 238</a> )	Waits until the module's measurement completes.



**Properties**

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

**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**

Property	Description
_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 256</a> )	Gets the number of modules in the collection.
Item (see <a href="#">page 266</a> )	Gets one of the modules in a collection given either a slot, index, or name.

**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
```

**Methods**

Method	Description
GetLine (see <a href="#">page 194</a> )	Gets an instruction or vector at line number.
GetLineLabel (see <a href="#">page 196</a> )	Gets a vector's label value at line number.
InsertLine (see <a href="#">page 214</a> )	Inserts a new instruction or vector after line number.
RemoveLine (see <a href="#">page 226</a> )	Removes the instruction or vector at line number.
Reset (see <a href="#">page 226</a> )	Resets the current line number to the first line.
Resume (see <a href="#">page 227</a> )	Resumes running the pattern generator from the current line number.
Run (see <a href="#">page 228</a> )	Starts running the pattern generator.
SetLine (see <a href="#">page 230</a> )	Sets an instruction or vector at line number.
SetLineLabel (see <a href="#">page 230</a> )	Sets a vector's label value at line number.
Step (see <a href="#">page 235</a> )	Steps the pattern generator from the current line number.
Stop (see <a href="#">page 236</a> )	Stops the pattern generator if it is currently running.

(Also Includes Module (see [page 128](#)) object methods)

### Properties

Property	Description
NumLines (see <a href="#">page 270</a> )	Gets the number of lines in the main sequence.

(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
// 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::IIstrumentPtr 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 = strlen(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

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 136](#)) ]

- To Access**
- Instrument.**GetProbeByName(Name)** (see [page 200](#))
  - Probes.**Item** (see [page 266](#)) IndexOrName
  - Probes IndexOrName

### Methods

Method	Description
DoAction (see <a href="#">page 174</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 174</a> )	Execute a particular XML-based command.
QueryCommand (see <a href="#">page 220</a> )	Query for XML-based commands.

### Properties

Property	Description
Name (see <a href="#">page 269</a> )	Gets or sets the name of the probe.
Type (see <a href="#">page 285</a> )	Gets the probe's type.

- See Also**
- Probes (see [page 135](#)) object

## Probes Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 136](#)) ]

- To Access**
- Instrument.**Probes** (see [page 274](#))

**Methods** There are no methods.

### Properties

Property	Description
_NewEnum (see <a href="#">page 289</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 256</a> )	Gets the number of probes in the collection.
Item (see <a href="#">page 266</a> )	Gets one of the probes in the collection given either an index or name.

- 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 + vbNewLine + 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);
```



```

////////////////////////////////////
//
//  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::IIInstrumentPtr 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;
}

```

```

////////////////////////////////////
//
// 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);
}

```

## 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

```

## Methods

| Method                                             | Description                                                                                         |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| GetDataBySample (see <a href="#">page 189</a> )    | Given a sample range, returns an array of data.                                                     |
| GetDataByTime (see <a href="#">page 192</a> )      | Given a time range, returns an array of data.                                                       |
| GetNumSamples (see <a href="#">page 199</a> )      | Given a range, returns the number of samples stored.                                                |
| GetSampleNumByTime (see <a href="#">page 204</a> ) | Gets the closest sample number corresponding to the time given.                                     |
| GetTime (see <a href="#">page 204</a> )            | Given a range, returns the time for this bus/signal in the format specified by the data type given. |

## Properties

| Property                                    | Description                                                 |
|---------------------------------------------|-------------------------------------------------------------|
| DataType (see <a href="#">page 260</a> )    | Gets the recommended bus/signal data type.                  |
| EndSample (see <a href="#">page 261</a> )   | Gets the data's ending sample number relative to trigger.   |
| EndTime (see <a href="#">page 262</a> )     | Gets the data's ending time relative to trigger.            |
| StartSample (see <a href="#">page 279</a> ) | Gets the data's starting sample number relative to trigger. |
| StartTime (see <a href="#">page 279</a> )   | Gets the data's starting time relative to trigger.          |

## 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) + _
            vbNewLine
        printHeader = False
    End If

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

    ' 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 \
// 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);

```

```

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::IIInstrumentPtr 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' ");
        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);
}
}

```

```

        }

        default:
            break;

    }
}

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

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

return 0;
}

//
//
// 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);

```



```

        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]: %01e\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 = strlen(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

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

**To Access**

- SampleDifferences.Item (see [page 266](#)) IndexOrName
- SampleDifferences IndexOrName

**Methods** There are no methods.

### Properties

| Property                                             | Description                                                                  |
|------------------------------------------------------|------------------------------------------------------------------------------|
| BusSignalDifferences (see <a href="#">page 250</a> ) | Gets a collection of all the buses/signals with differences for this sample. |
| SampleNum (see <a href="#">page 277</a> )            | Gets the sample number at which differences occurred.                        |

**See Also**

- SampleDifferences (see [page 149](#)) object

## SampleDifferences Object

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

**To Access**

- CompareWindow.SampleDifferences (see [page 276](#))

**Methods** There are no methods.

### Properties

| Property                                 | Description                                                                                                             |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| _NewEnum (see <a href="#">page 289</a> ) | Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .                                        |
| Count (see <a href="#">page 256</a> )    | Gets the number of bus/signal differences in the collection.                                                            |
| Item (see <a href="#">page 266</a> )     | Given an index into the collection, gets a SampleDifference (see <a href="#">page 149</a> ) object from the collection. |

**See Also**

- SampleDifferences (see [page 276](#)) property

## SelfTest Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 150](#)) ]

**To Access** • Instrument.**SelfTest** (see [page 277](#))

**Methods**

| Method                                  | Description                      |
|-----------------------------------------|----------------------------------|
| TestAll (see <a href="#">page 237</a> ) | Runs an instrument's self-tests. |

**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**

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

**Visual C++**

```
//
// 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"

////////////////////////////////////
//
// 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 = "col-mil20"; // 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::IIInstrumentPtr 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\n", (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 = strlen(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

| Method                                       | Description                                  |
|----------------------------------------------|----------------------------------------------|
| DoAction (see <a href="#">page 174</a> )     | Execute a specific XML-based command action. |
| DoCommands (see <a href="#">page 174</a> )   | Execute a particular XML-based command.      |
| QueryCommand (see <a href="#">page 220</a> ) | Query for XML-based commands.                |

### Properties

| Property                                   | Description                                          |
|--------------------------------------------|------------------------------------------------------|
| BusSignals (see <a href="#">page 250</a> ) | Gets a collection of the tool's defined bus/signals. |
| Name (see <a href="#">page 269</a> )       | Gets or sets the name of the tool.                   |
| Type (see <a href="#">page 285</a> )       | Gets the specific tool type.                         |

- 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

| Property                                 | Description                                                                      |
|------------------------------------------|----------------------------------------------------------------------------------|
| _NewEnum (see <a href="#">page 289</a> ) | Used by Visual Basic to support the implementation of <b>For Each ... Next</b> . |
| Count (see <a href="#">page 256</a> )    | Gets the number of tools in the collection.                                      |
| Item (see <a href="#">page 266</a> )     | Gets one of the tools in the collection given either an index or name.           |

**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 + vbNewLine + 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
```

**Visual C++**

```

        MsgBox "MPC8XX IA Properties: " + XMLCmdResponse
    End If

//
// 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
// 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 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);
}
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 = strlen(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

| Method                               | Description      |
|--------------------------------------|------------------|
| Draw (see <a href="#">page 178</a> ) | Draws the chart. |

### Properties

| Property                                  | Description                             |
|-------------------------------------------|-----------------------------------------|
| Axis (see <a href="#">page 246</a> )      | Gets the chart axis given an axis type. |
| ChartType (see <a href="#">page 253</a> ) | Gets or sets the chart type.            |
| Data (see <a href="#">page 260</a> )      | Gets the chart data.                    |
| HasLegend (see <a href="#">page 264</a> ) | Gets or sets if the legend is visible.  |
| HasTitle (see <a href="#">page 264</a> )  | Gets or sets if the title is visible.   |
| Legend (see <a href="#">page 267</a> )    | Gets the chart legend.                  |
| Title (see <a href="#">page 284</a> )     | Gets the title of the chart.            |

**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

| Property                                 | Description                                                                         |
|------------------------------------------|-------------------------------------------------------------------------------------|
| AxisBase (see <a href="#">page 246</a> ) | Gets or sets the chart axis base.                                                   |
| BitSize (see <a href="#">page 248</a> )  | Gets or sets the width of the data in bits. This is used to format the Axis values. |
| HasTitle (see <a href="#">page 264</a> ) | Gets or sets if the title is visible.                                               |
| Title (see <a href="#">page 284</a> )    | Gets the title of the axis.                                                         |

**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

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

| Method                                          | Description                                                     |
|-------------------------------------------------|-----------------------------------------------------------------|
| SetValueArray (see <a href="#">page 231</a> )   | Sets an array of values in the chart array starting at index 0. |
| SetValueCaption (see <a href="#">page 232</a> ) | Sets the caption associated with all values at index (column).  |

**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**

| Property                                 | Description                             |
|------------------------------------------|-----------------------------------------|
| Bold (see <a href="#">page 248</a> )     | Gets or sets the text thickness.        |
| Color (see <a href="#">page 254</a> )    | Gets or sets the text color.            |
| FaceName (see <a href="#">page 262</a> ) | Gets or sets the text face name string. |
| Size (see <a href="#">page 278</a> )     | Gets or sets the text size.             |

**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**

| Property                                 | Description                             |
|------------------------------------------|-----------------------------------------|
| Position (see <a href="#">page 273</a> ) | Gets or sets the chart legend position. |

**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

| Property                                | Description                           |
|-----------------------------------------|---------------------------------------|
| Caption (see <a href="#">page 251</a> ) | Gets or sets the chart title caption. |
| Font (see <a href="#">page 262</a> )    | Gets the chart title font.            |

**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

| Method                                | Description                 |
|---------------------------------------|-----------------------------|
| Clear (see <a href="#">page 171</a> ) | Displays an empty web page. |

### Properties

| Property                                   | Description                                |
|--------------------------------------------|--------------------------------------------|
| WebBrowser (see <a href="#">page 288</a> ) | Gets the contained IWebBrowser2 interface. |

**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

| Method                                      | Description                                |
|---------------------------------------------|--------------------------------------------|
| ClearOutput (see <a href="#">page 172</a> ) | Clears the strings from the output window. |
| WriteOutput (see <a href="#">page 242</a> ) | Writes a string to the output window.      |

(Also Includes Window (see [page 160](#)) methods)

### Properties

| Property                                   | Description                |
|--------------------------------------------|----------------------------|
| Chart (see <a href="#">page 253</a> )      | Gets the Chart view.       |
| WebBrowser (see <a href="#">page 288</a> ) | Gets the Web Browser view. |

(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

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

### Properties

| Property                                                | Description                                            |
|---------------------------------------------------------|--------------------------------------------------------|
| <code>BusSignals</code> (see <a href="#">page 250</a> ) | Gets a collection of the window's defined bus/signals. |
| <code>Name</code> (see <a href="#">page 269</a> )       | Gets or sets the name of the window.                   |
| <code>Type</code> (see <a href="#">page 285</a> )       | Gets the window's type.                                |

- 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**

| Property                                 | Description                                                                      |
|------------------------------------------|----------------------------------------------------------------------------------|
| _NewEnum (see <a href="#">page 289</a> ) | Used by Visual Basic to support the implementation of <b>For Each ... Next</b> . |
| Count (see <a href="#">page 256</a> )    | Gets the number of windows in the collection.                                    |
| Item (see <a href="#">page 266</a> )     | Gets one of the windows in the collection given either an index or name.         |

- 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
                                     // 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::IIInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

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

            // Display all of the window names.
            AgtLA::IWindowsPtr pWindows = pInst->GetWindows();
            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' \
        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 = strlen(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](#))

- GetToolByName 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** object.**Add** Name, Channels, [Polarity="+"]

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>BusSignals</b> (see <a href="#">page 113</a> ) object. |
| Name       | A <b>String</b> containing the name of the bus/signal to be added.                          |



| Parameters | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Channels   | <p>A <b>String</b> containing <i>MultiplePodChannels</i> or the <b>String</b> "None" if no channels are assigned.</p> <p><b>MultiplePodChannels</b> contains a comma separated list of <i>PodChannels</i>.</p> <p><b>PodChannels</b> contains a <i>PodNumber</i> followed by the <b>String</b> "[" followed by a comma separated list of individual channel <i>Number(s)</i> and <i>NumberRange(s)</i> followed by the <b>String</b> "]".</p> <p><b>Note:</b> Pod channels normally are in MSB to LSB notation unless you are trying to reorder channels.</p> <p><b>NumberRange</b> contains a <i>Number</i> followed by the <b>String</b> ":" followed by a <i>Number</i>.</p> <p>PodChannels Example:<br/> Pod 1 [ 9 : 7 , 5 , 3 : 1 ] — this bus consists of 1 single channel <i>Number</i> and 2 <i>NumberRange</i>'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].</p> <p>MultiplePodChannels Example:<br/> Pod 2 [ 3 , 1 ] , Pod 3 [ 10 , 5 : 3 ] — this bus consists of 3 single channel <i>Numbers</i> and 1 <i>NumberRange</i> for a total of 6 channels Pod 2[3], Pod 2[1], Pod 3[10], Pod 3[5], Pod 3[4], Pod 3[3].</p> |
| Polarity   | A <b>String</b> that is either "+" or "-". This parameter is optional and is "+" if not specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

## 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

| Parameters      | Definition                                                                                      |
|-----------------|-------------------------------------------------------------------------------------------------|
| object          | An expression that evaluates to a <b>Markers</b> (see <a href="#">page 123</a> ) object.        |
| Name            | A <b>String</b> containing the name of the marker to be added.                                  |
| TextColor       | A <b>Long</b> representing the color of the text.                                               |
| BackgroundColor | A <b>Long</b> representing the color of the background.                                         |
| Position        | A <b>Double</b> that specifies the position of the marker (in seconds) relative to the trigger. |

## 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

| Parameters | Definition                                                                                                                       |
|------------|----------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>Markers</b> (see <a href="#">page 123</a> ) object.                                         |
| XMLMarkers | An "XML format" (in the online help) <b>String</b> 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]

| Parameters  | Definition                                                                                                          |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| object      | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object.                          |
| XValueArray | An array of <b>Variants</b> that are the X values to set in the chart.                                              |
| YValueArray | An array of <b>Variants</b> that are the Y values to set in the chart.                                              |
| PointType   | An <b>AgtDataPointType</b> enumerated type value that specifies the shape of the point. See the descriptions below. |
| PointSize   | An <b>AgtDataPointSize</b> enumerated type value that specifies the size of the point. See the descriptions below.  |

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

| AgtDataPointType     | Enum Value | Description |
|----------------------|------------|-------------|
| AgtDataPointTypeNone | 1          |             |

|                                |   |  |
|--------------------------------|---|--|
| AgtDataPointTypeAutomatic      | 2 |  |
| AgtDataPointTypeSquare         | 3 |  |
| AgtDataPointTypeDiamond        | 4 |  |
| AgtDataPointTypeTriangle       | 5 |  |
| AgtDataPointTypeInvertTriangle | 6 |  |
| AgtDataPointTypeStar           | 7 |  |
| AgtDataPointTypeCircle         | 8 |  |

The **PointSize** parameter can have the following values:

| <b>AgtDataPointSize</b> | <b>Enum Value</b> | <b>Description</b> |
|-------------------------|-------------------|--------------------|
| AgtDataPointSizeSmall   | 1                 |                    |
| AgtDataPointSizeMedium  | 2                 |                    |
| AgtDataPointSizeLarge   | 3                 |                    |

### 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**

| <b>Parameters</b> | <b>Definition</b>                                                                          |
|-------------------|--------------------------------------------------------------------------------------------|
| object            | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |

### 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**

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWebBrowser (see <a href="#">page 159</a> ) object. |

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

## 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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWindow (see <a href="#">page 160</a> ) object. |

## 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**]

| Parameters   | Definition                                                                                                                                                                                                  |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object       | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                                       |
| SaveChanges  | A <b>Boolean</b> that specifies whether changes to the configuration should be saved.                                                                                                                       |
| SaveFileName | A <b>String</b> that is the name of the file to which the configuration information should be saved.                                                                                                        |
| SetupOnly    | A <b>Boolean</b> that specifies whether the configuration file contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

## Connect Method

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

**Applies To** • **ConnectSystem** (see [page 119](#)) object

**Description** Connects to the remote logic analyzer system.

**VB Syntax** `object.Connect [HostNameOrIpAddress=""]`

| Parameters          | Definition                                                                                                                                                                                         |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object              | An expression that evaluates to a <b>ConnectSystem</b> (see <a href="#">page 119</a> ) object.                                                                                                     |
| HostNameOrIpAddress | A <b>String</b> that contains the hostname or IP address of the logic analyzer instrument or computer on which the <i>Agilent Logic Analyzer</i> application will run. This parameter is optional. |

**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`

| Parameters   | Definition                                                                               |
|--------------|------------------------------------------------------------------------------------------|
| object       | An expression that evaluates to a <b>Connect</b> (see <a href="#">page 118</a> ) object. |
| SrcFileName  | A <b>String</b> that is the name of the file on the local file system.                   |
| DestFileName | A <b>String</b> that is the name of the file on the instrument file system.              |

**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`

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |
| FileName   | A <b>String</b> that is the name of the file on the instrument file system.           |

## DoAction Method

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

- 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

| Parameters | Definition                                                                                                                                                               |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above.                                                                                       |
| Action     | Name of the command to execute. For information about the XML-based actions supported by a tool, see the "Tool Control, COM Automation" topic in the tool's online help. |
| Parameters | Command parameters.                                                                                                                                                      |

**Return Value** A **Boolean** indicating whether the command was successful.

## DoCommands Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 175](#)) ]

- 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.

**VB Syntax**    `object.DoCommands XMLCommand`

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above.   |
| XMLCommand | An XML-format string that configures a module, tool, or window. See "Remarks" below. |

**Return Value**    A **Boolean** indicating whether the command was successful.

**Remarks**        The XMLCommand format is based on the object type:

| Object                                     | XMLCommand Format                                                                                       |
|--------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Instrument (see <a href="#">page 121</a> ) | A string containing the XML format "<Configuration> element" (in the online help).                      |
| Module (see <a href="#">page 128</a> )     | A string containing the XML format "<Module> element" (in the online help) (under Configuration Setup). |
| Probe (see <a href="#">page 135</a> )      | A string containing the XML format "<Probe> element" (in the online help) (under Configuration Setup).  |
| Tool (see <a href="#">page 152</a> )       | A string containing the XML format "<Tool> element" (in the online help) (under Configuration Setup).   |
| Window (see <a href="#">page 160</a> )     | A string containing the XML format "<Window> element" (in the online help) (under Configuration Setup). |

## 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
```

```

        MsgBox "MPC8XX IA Properties: " + XMLCommand
    End If

    ' Execute XML-based command.
    If myTool.DoCommands(XMLCommand) Then
        MsgBox "MPC8XX IA DoCommands successful."
    End If

```

**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::IIInstrumentPtr 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;
}

////////////////////////////////////
//
// 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);
}

```

## 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**

| Parameters | Definition                                                                             |
|------------|----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChart (see <a href="#">page 156</a> ) object. |

## 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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a CompareWindow (see <a href="#">page 118</a> ) object. |

- 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]

| Parameters          | Definition                                                                                                                                                                 |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object              | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                      |
| ExportFileName      | A <b>String</b> that contains the name of the file (on the instrument file system) to which data is exported.                                                              |
| SourceName          | A <b>String</b> that contains the name of the Module, Tool, or Window whose data will be exported.                                                                         |
| ExportRange         | A <b>Boolean</b> that specifies whether a data range is used: <b>True</b> – StartRange and EndRange contain the data range to export. <b>False</b> – exports all the data. |
| StartRange EndRange | <b>Variants</b> specifying the data range (see <a href="#">page 180</a> ).                                                                                                 |

**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.

| For:           | Use the Variant Type:         |
|----------------|-------------------------------|
| sample numbers | <b>Integer</b> or <b>Long</b> |
| times          | <b>Double</b>                 |

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=""]

| Parameters     | Definition                                                                                                    |
|----------------|---------------------------------------------------------------------------------------------------------------|
| object         | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                         |
| ExportFileName | A <b>String</b> that contains the name of the file (on the instrument file system) to which data is exported. |

| Parameters          | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SourceName          | <p>A <b>String</b> that contains the name of the Module, Tool, or Window whose data will be exported.</p> <p>When the "Module CSV text file" or "Module binary file" FileType is specified, the SourceName must be a logic analyzer or import module. Timing zoom data can be exported separately using the SourceName syntax of "&lt;module_name&gt;:TimingZoom", for example, "My 16950A-1:TimingZoom".</p> <p>When the "Pattern Generator CSV text file" FileType is specified, the SourceName must be a pattern generator module.</p> |
| ExportRange         | <p>A <b>Boolean</b> that specifies whether a data range is used: <b>True</b> – StartRange and EndRange contain the data range to export. <b>False</b> – exports all the data.</p>                                                                                                                                                                                                                                                                                                                                                         |
| StartRange EndRange | <p><b>Variants</b> specifying the data range (see <a href="#">page 180</a>).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

| Parameters          | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SourceName          | <p>A <b>String</b> that contains the name of the Module, Tool, or Window whose data will be exported.</p> <p>When the "Module CSV text file" or "Module binary file" FileType is specified, the SourceName must be a logic analyzer or import module. Timing zoom data can be exported separately using the SourceName syntax of "&lt;module_name&gt;:TimingZoom", for example, "My 16950A-1:TimingZoom".</p> <p>When the "Pattern Generator CSV text file" FileType is specified, the SourceName must be a pattern generator module.</p> |
| ExportRange         | <p>A <b>Boolean</b> that specifies whether a data range is used: <b>True</b> – StartRange and EndRange contain the data range to export. <b>False</b> – exports all the data.</p>                                                                                                                                                                                                                                                                                                                                                         |
| StartRange EndRange | <p><b>Variants</b> specifying the data range (see <a href="#">page 180</a>).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

| Parameters  | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FileType    | <p>A <b>String</b> that identifies the type of data you want to export. This is the same string that you see in the <i>Agilent Logic Analyzer</i> application's Export dialog:</p> <ul style="list-style-type: none"> <li>• "Standard CSV text file"</li> <li>• "Module CSV text file"</li> <li>• "Module binary file"</li> <li>• "Pattern Generator CSV text file"</li> <li>• "16700 ASCII File Format"</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| FileOptions | <p>An XML format <b>String</b> that lets you specify export options (as you can with the <i>Agilent Logic Analyzer</i> application's Export dialog <b>Options...</b> button). For example:</p> <pre>&lt;Options   SeparationCharacters=', '   WriteLineNumberColumn='T'   LineNumberColumnName='Line Number'   IncludeHeader='T'   WriteFixedWidthColumns='F' /&gt;</pre> <p>The &lt;Options&gt; element attribute values can be:</p> <ul style="list-style-type: none"> <li>• SeparationCharacters — 'string'</li> <li>• WriteLineNumberColumn — 'F' (false) or 'T' (true)</li> <li>• LineNumberColumnName — 'string'</li> <li>• IncludeHeader — 'F' (false) or 'T' (true)</li> <li>• WriteFixedWidthColumns — 'F' (false) or 'T' (true)</li> </ul> <p>Certain file types support certain file options (as in the <i>Agilent Logic Analyzer</i> application):</p> <ul style="list-style-type: none"> <li>• "Standard CSV text file" — SeparationCharacters, WriteLineNumberColumns, and LineNumberColumnName.</li> <li>• "Module CSV text file" — SeparationCharacters, WriteLineNumberColumns, LineNumberColumnName, and IncludeHeader.</li> <li>• "Module binary file" — IncludeHeader.</li> <li>• "Pattern Generator CSV text file" — SeparationCharacters.</li> <li>• "16700 ASCII File Format" — (none).</li> </ul> |

**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

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

**Applies To** • **Window** (see [page 160](#)) object

**Description** Finds a specified data event with optional occurrence and time duration.

**VB Syntax** object.**Find** Event, [Occurrence=1], [Direction="F"], [From="Display Center"], [When="Present"], [Duration=""]

| Parameters | Definition                                                                                                                                                                                                                                                                                                                                 |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Window (see <a href="#">page 160</a> ) object.                                                                                                                                                                                                                                                           |
| Event      | A <b>String</b> containing the event to find. The event can be a simple string (see Event (see <a href="#">page 233</a> )) or, for more complex searches, an XML format string (see "<Event> element (under Find)" (in the online help)).                                                                                                  |
| Occurrence | A <b>Long</b> containing the number of occurrences of the Event parameter.                                                                                                                                                                                                                                                                 |
| Direction  | A <b>String</b> containing the direction to search. "F" searches forward, "B" searches backward.                                                                                                                                                                                                                                           |
| From       | A <b>String</b> containing the position to start searching. This can be any of the following: <ul style="list-style-type: none"> <li>• "Display Center"</li> <li>• "Beginning Of Data"</li> <li>• "End Of Data"</li> <li>• "Trigger"</li> <li>• Name of a currently defined marker.</li> </ul> Note that these strings are case-sensitive. |



| Parameters | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| When       | <p>A <b>String</b> specifying a time duration or other operator.</p> <ul style="list-style-type: none"> <li>• "Present"</li> <li>• "Not Present"</li> <li>• "Present&gt;" (Duration must contain only one time value)</li> <li>• "Present&gt;=" (Duration must contain only one time value)</li> <li>• "Present&lt;" (Duration must contain only one time value)</li> <li>• "Present&lt;=" (Duration must contain only one time value)</li> <li>• "Present for Range" (Duration must contain a time range)</li> <li>• "Not In Range" (Duration must contain a time range)</li> <li>• "Entering"</li> <li>• "Exiting"</li> <li>• "Transitioning"</li> </ul> <p>Note that these strings are case-sensitive.</p>   |
| Duration   | <p>A <b>String</b> 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:</p> <ul style="list-style-type: none"> <li>• "ps" - picoseconds</li> <li>• "ns" - nanoseconds</li> <li>• "us" - microseconds</li> <li>• "ms" - milliseconds</li> <li>• "s" - seconds</li> <li>• "Gs" - gigaseconds</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• "1 ps"</li> <li>• "1ns..30ns"</li> </ul> |

**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**

```

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 \
// 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::IIstrumentPtr 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 = strlen(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**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Window (see <a href="#">page 160</a> ) object. |

**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

## FindPrev Method

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

**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**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Window (see <a href="#">page 160</a> ) object. |

**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

## GetDataBySample 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 an array of acquired data. **GetDataBySample** returns the data within a trigger relative sample 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.GetDataBySample StartSample, EndSample, DataType, NumRowsRet`

| Parameters  | Definition                                                                                                  |
|-------------|-------------------------------------------------------------------------------------------------------------|
| object      | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object.              |
| StartSample | A <b>Long</b> containing the first sample to upload.                                                        |
| EndSample   | A <b>Long</b> containing the last sample to upload. EndSample must be greater than or equal to StartSample. |
| DataType    | Specifies the type of data to return. See DataTypes and Return Values (see <a href="#">page 190</a> ).      |

| Returns    | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| NumRowsRet | A <b>Long</b> initialized by this method to the number of rows being returned in the array. |

**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

| AgtDataType | Enum Value | Return Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AgtDataRow  | &H0001 (1) | <p>Returns an array of <b>Bytes</b>. The total size of the array is NumRowsRet multiplied by the value in the BusSignal (see <a href="#">page 108</a>) object's ByteSize (see <a href="#">page 251</a>) property. This can hold the maximum BusSignal size of 128 bits of unsigned data.</p> <p>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.</p> |

|                  |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AgtDataDecimal   | &H0002 (2) | Returns an array of <b>Variants</b> . The <b>Variant</b> 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 BusSignalType (see <a href="#">page 249</a> ) property is <b>AgtBusSignalSampleNum</b> , when the GetTime (see <a href="#">page 204</a> ) 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. |
| AgtDataLong      | &H0003 (3) | Returns an array of <b>Longs</b> . This holds 31 bits of unsigned integer data and 32 bits of signed integer data. This data type can be used when the BusSignalType (see <a href="#">page 249</a> ) property is <b>AgtBusSignalSampleNum</b> 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.                                                                                                                                                                                                                                                                                                                                                              |
| AgtDataTime      | &H0004 (4) | Returns an array of <b>Doubles</b> . This data type is only valid when the GetTime (see <a href="#">page 204</a> ) method is called. You can also use <b>AgtDataDouble</b> and <b>AgtDataDecimal</b> to access time values as well.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| AgtDataStringDec | &H0005 (5) | Returns an array of <b>Strings</b> , formatted in decimal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| AgtDataStringHex | &H0006 (6) | Returns an array of <b>Strings</b> , formatted in hexadecimal. When the GetTime (see <a href="#">page 204</a> ) 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.                                                                                                                                                                                                                                                                                                       |

|               |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AgtDataString | &H0007 (7) | Returns an array of <b>Strings</b> using the default format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| AgtDataDouble | &H0008 (8) | Returns an array of <b>Doubles</b> . 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 <a href="#">page 249</a> ) property is <b>AgtBusSignalSampleNum</b> , when the GetTime (see <a href="#">page 204</a> ) 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. |

| OR'ed in AgtDataType | Enum Value   | Return Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AgtDataSubrows       | &H0100 (256) | <p>Returns an array of arrays of type specified by the lower 8 bits of <b>EnumValue</b>.</p> <p>For example, if the <b>EnumValue</b> 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 <a href="#">page 249</a>) property is <b>AgtBusSignalGenerated</b>, there are cases when the array may be empty.</p> <pre>sArray() = GetDataBySample(0,10, AgtDataString Or AgtDataSubrows, nNumRowsRet) ' Visual Basic sArray = GetDataBySample(0,10, AgtDataString   AgtDataSubrows, nNumRowsRet); /* C/C++ */</pre> |
| AgtDataSigned        | &H0200 (512) | <p>Returns signed values of type specified by the lower 8 bits of <b>EnumValue</b>. This value is ignored for <b>EnumValueAgtDataRaw</b>.</p> <p>For example, if the <b>EnumValue</b> is a bitwise OR of AgtDataDouble and AgtDataSigned, each sample will be converted to a signed value and returned as an array of doubles.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## GetDataByTime 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 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

| Parameters | Definition                                                                                                                                                                                                                                                           |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object.                                                                                                                                                                       |
| StartTime  | A <b>Double</b> containing the starting time (in seconds) to upload. <b>Double</b> 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. |
| EndTime    | A <b>Double</b> containing the ending time (in seconds) to upload. <b>EndTime</b> must be greater than or equal to <b>StartTime</b> .                                                                                                                                |
| DataType   | Specifies the type of data to return. See DataTypes and Return Values (see <a href="#">page 190</a> ).                                                                                                                                                               |

| Returns    | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| NumRowsRet | A <b>Long</b> initialized by this method to the number of rows being returned in the array. |

**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

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Group      | A <b>Long</b> representing the group (row) on which to set the caption.                    |

**Return Value** A **String** containing the group caption.

**See Also** • SetGroupCaption (see [page 229](#)) method

## 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

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |
| LineNumber | A <b>Long</b> representing the line number from which to get the instruction or vector. |

**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:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <label_value>     | '<label_name>' = <value>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <parameter_value> | '<parameter_name>' = <value>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <wait_pattern>    | A 3-digit binary number as in the user-interface (for example, 001).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <module_name>     | The name of a module in the frame.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <value>           | <p>Values are formatted consistently with XML and COM number formatting. Numbers are case-insensitive and must contain a number base prefix:</p> <ul style="list-style-type: none"> <li>• h — hexadecimal</li> <li>• o — octal</li> <li>• b — binary</li> <li>• d — decimal</li> </ul> <p>Numbers can optionally contain don't care symbols:</p> <ul style="list-style-type: none"> <li>• X — don't care</li> </ul> <p>Some example values are:</p> <ul style="list-style-type: none"> <li>• hfx</li> <li>• o72</li> <li>• b11110000</li> <li>• d24</li> <li>• b1111xxxx</li> </ul> |

#### 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

- Main End
- End Loop

- 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

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Gets a vector's label value at line number.

**VB Syntax** object.**GetLineLabel** LineNumber, LabelName

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |
| LineNumber | A <b>Long</b> representing the line number from which to get the instruction or vector. |
| LabelName  | A <b>String</b> 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

## GetModuleByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 197](#)) ]

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Given a module name, returns its corresponding hardware module object.

**VB Syntax** `object.GetModuleByName Name`

| Parameters | Definition                                                                                                    |
|------------|---------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                         |
| Name       | A <b>String</b> containing the module's name as defined by the Name (see <a href="#">page 269</a> ) property. |

**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++**

```
//
// 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
```

```

// 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::IIInstrumentPtr pInst =
                pConnect->GetInstrument(hostname);

            // Get a specific analyzer module.
            _bstr_t moduleName = "My 16910A-1";
            AgtLA::IIAnalyzerModulePtr pAnalyzer =
                pInst->GetModuleByName(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 = strlen(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]`

| Parameters                | Definition                                                                                                                                            |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| object                    | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object.                                                        |
| StartPosition EndPosition | <b>Variants</b> specifying the data range (see <a href="#">page 180</a> ). <b>EndPosition</b> must be greater than or equal to <b>StartPosition</b> . |

**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`

| Parameters | Definition                                                                                                   |
|------------|--------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                        |
| Name       | A <b>String</b> containing the probe's name as defined by the Name (see <a href="#">page 269</a> ) property. |

**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 <a href="#">page 106</a> ) object.                                                             |
| StartPosition EndPosition | <b>Variants</b> specifying the data range (see <a href="#">page 180</a> ). <b>EndPosition</b> must be greater than or equal to <b>StartPosition</b> . |

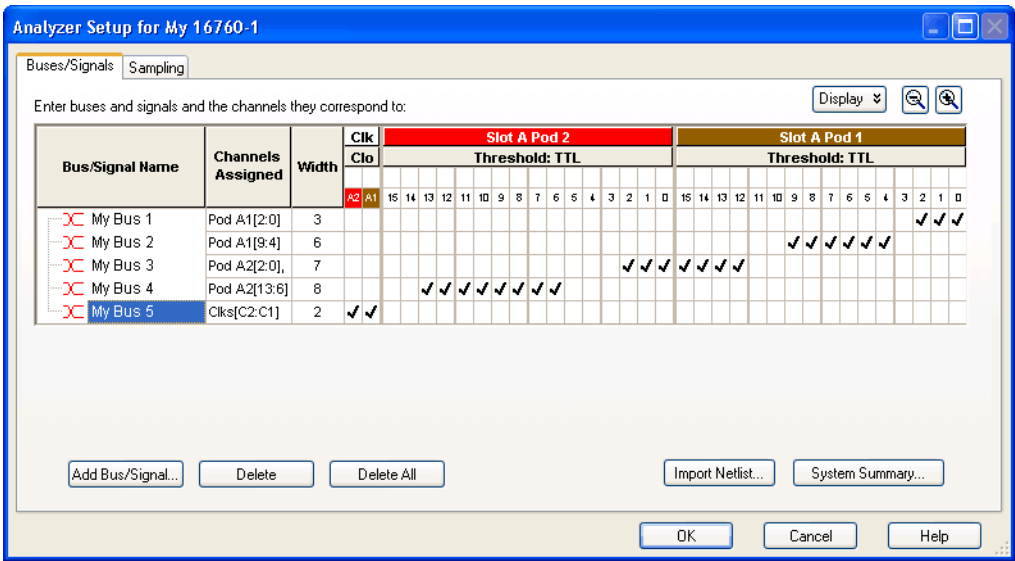
| Returns        | Definition                                                                                            |
|----------------|-------------------------------------------------------------------------------------------------------|
| NumBytesPerRow | A <b>Long</b> initialized by this method to the width of each sample row being returned in the array. |
| NumRowsRet     | A <b>Long</b> 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     |
| -----    | -----    | -----    | -----    | -----    |
| array[0] | array[1] | array[2] | array[3] | array[4] |

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

[ [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 timing zoom 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.GetRawTimingZoomData StartPosition, EndPosition, NumBytesPerRow, NumRowsRet`

| Parameters                | Definition                                                                                                                                            |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| object                    | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object.                                                             |
| StartPosition EndPosition | <b>Variants</b> specifying the data range (see <a href="#">page 180</a> ). <b>EndPosition</b> must be greater than or equal to <b>StartPosition</b> . |

| Returns        | Definition                                                                                            |
|----------------|-------------------------------------------------------------------------------------------------------|
| NumBytesPerRow | A <b>Long</b> initialized by this method to the width of each sample row being returned in the array. |
| NumRowsRet     | A <b>Long</b> initialized by this method to the number of rows being returned in the array.           |

**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** object.**GetRemoteInfo** HostNameOrIPAddress, RemoteUserName, RemoteComputerName

| Parameters          | Definition                                                                        |
|---------------------|-----------------------------------------------------------------------------------|
| object              | An expression that evaluates to a Connect (see <a href="#">page 118</a> ) object. |
| HostNameOrIPAddress | A <b>String</b> that is the hostname or IP address of the logic analyzer.         |

| Returns            | Definition                                          |
|--------------------|-----------------------------------------------------|
| RemoteUserName     | A <b>String</b> that is the remote user login name. |
| RemoteComputerName | A <b>String</b> that is the remote computer name.   |

- See Also**
- RemoteComputerName (see [page 274](#)) property
  - RemoteUserName (see [page 275](#)) property

## GetSampleNumByTime Method

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

- Applies To**
- **SampleBusSignalData** (see [page 138](#)) object

**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** (see [page 238](#)) method to make sure the hardware is stopped.

**VB Syntax**    object.**GetSampleNumByTime** Time

| Parameters | Definition                                                                                                                                                                                                                                                                    |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object.                                                                                                                                                                                |
| Time       | A <b>Double</b> containing the time that you want to get the sample number for.<br><b>Double</b> 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 Time value of 450E-9 is 450 nanoseconds. |

**Return Values**    A **Long** containing the sample number.

## GetTime 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 time for this bus/signal in the format specified by the data type given. **GetTime** gets the time values for the specific bus/signal.

**NOTE**

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**    object.**GetTime** StartPosition, EndPosition, DataType, NumRowsRet

| Parameters                | Definition                                                                                                                                            |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| object                    | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object.                                                        |
| StartPosition EndPosition | <b>Variants</b> specifying the data range (see <a href="#">page 180</a> ). <b>EndPosition</b> must be greater than or equal to <b>StartPosition</b> . |
| DataType                  | Specifies the type of data to return. See DataTypes and Return Values (see <a href="#">page 190</a> ).                                                |

| Returns    | Definition                                                                                          |
|------------|-----------------------------------------------------------------------------------------------------|
| NumRowsRet | A <b>Variant</b> initialized by this method to the number of rows being returned in the time array. |

**Return Values**    See DataTypes and Return Values (see [page 190](#)).

- See Also**
- GetDataByTime (see [page 192](#)) method
  - GetDataBySample (see [page 189](#)) method

## GetToolByName Method

[ 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**    object.**GetToolByName** Name

| Parameters | Definition                                                                                                  |
|------------|-------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                       |
| Name       | A <b>String</b> containing the tool's name as defined by the Name (see <a href="#">page 269</a> ) property. |

**Return Value**    A Tool (see [page 152](#)) object with the tool name given.

- See Also**
- Tools (see [page 284](#)) property

- Name (see [page 269](#)) property

## GetValueCaption Method

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

**Applies To** • **VbaViewChartData** (see [page 157](#)) object

**Description** Gets the caption associated with all values at index (column).

**VB Syntax** object.GetValueCaption Index

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Index      | A <b>Long</b> representing the index (column) from which to get the caption.               |

**Return Value** A **String** containing the value caption.

**See Also** • SetValueCaption (see [page 232](#)) method

## GetWindowByName Method

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

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Given a window name, returns the corresponding window object.

**VB Syntax** object.GetWindowByName Name

| Parameters | Definition                                                                                                    |
|------------|---------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                         |
| Name       | A <b>String</b> containing the window's name as defined by the Name (see <a href="#">page 269</a> ) property. |

**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                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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 \
// 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 is offline or online,
//  and if online the system it is 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(connectionTo);
}

//
//
//  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 = strlen(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** object.**GoOnline** [ComputerNameOrIPAddress=""] [FrameNumber=1] [SaveChanges=**False**] [SaveFileName=""] [SetupOnly=**False**]

| Parameters                | Definition                                                                                                                                                                                                                                     |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object                    | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                                                                          |
| ComputerNameOrIPAddresses | A <b>String</b> containing the frame's computer host name as defined by the ComputerName (see <a href="#">page 255</a> ) property or IP address as defined by the IPAddress (see <a href="#">page 265</a> ) property.                          |
| FrameNumber               | A <b>Long</b> that specifies the number of the frame to connect to in a multiframe configuration.                                                                                                                                              |
| SaveChanges               | A <b>Boolean</b> that specifies whether changes to the current configuration should be saved.                                                                                                                                                  |
| SaveFileName              | A <b>String</b> containing the name of the file to which current configuration information should be saved.                                                                                                                                    |
| SetupOnly                 | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

- See Also**
- GoOffline (see [page 207](#)) method
  - IsOnline (see [page 214](#)) method

## 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

| Parameters | Definition                                                                                                                                             |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Window (see <a href="#">page 160</a> ) object.                                                                       |
| Position   | A <b>Variant</b> that specifies the time of the sample to be placed at the center of the window. For time values, use the variant type <b>Double</b> . |

## 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**]

| Parameters     | Definition                                                                                                                                                                                                                                     |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object         | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                                                                          |
| ImportFileName | A <b>String</b> that contains the name of the file (on the instrument file system) to be imported.                                                                                                                                             |
| SaveChanges    | A <b>Boolean</b> that specifies whether changes to the current configuration should be saved.                                                                                                                                                  |
| SaveFileName   | A <b>String</b> containing the name of the file to which current configuration information should be saved.                                                                                                                                    |
| SetupOnly      | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

**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

[ 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 into a particular module.

**VB Syntax** object.**ImportEx** ImportFileName DestinationName [SaveChanges=**False**] [SaveFileName=""] [SetupOnly=**False**]

| Parameters      | Definition                                                                                                                                                                                                                                     |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object          | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                                                                          |
| ImportFileName  | A <b>String</b> that contains the name of the file (on the instrument file system) to be imported.                                                                                                                                             |
| DestinationName | A <b>String</b> that is the name of the module into which the data is imported.                                                                                                                                                                |
| SaveChanges     | A <b>Boolean</b> that specifies whether changes to the current configuration should be saved.                                                                                                                                                  |
| SaveFileName    | A <b>String</b> containing the name of the file to which current configuration information should be saved.                                                                                                                                    |
| SetupOnly       | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

**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.

- "Pattern Generator Binary file" (\*.pgb) – if the DestinationName is a pattern generator module.
- "16700 Fast Binary Data" – if DestinationName is empty.

**See Also**

- Import (see [page 212](#)) method
- Export (see [page 179](#)) method
- ExportEx (see [page 180](#)) method

## InsertLine Method

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Inserts a new instruction or vector after line number.

**VB Syntax** object.**InsertLine** LineNumber, InstructionOrVector

| Parameters          | Definition                                                                                               |
|---------------------|----------------------------------------------------------------------------------------------------------|
| object              | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object.                  |
| LineNumber          | A <b>Long</b> representing the line number after which the new instruction or vector should be inserted. |
| InstructionOrVector | A <b>String</b> containing the instruction or vector (see <a href="#">page 194</a> ) to be inserted.     |

**See Also** • RemoveLine (see [page 226](#)) method

## IsOnline Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 207](#)) ]

**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

| Parameters   | Definition                                                                                                                                                                                |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object       | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                     |
| ComputerName | A <b>String</b> that is initialized to the frame's name (as defined by the ComputerName (see <a href="#">page 255</a> ) property) if the user interface is online (connected to a frame). |

**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** object.**IsTimingZoom**

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an BusSignal (see <a href="#">page 108</a> ) object. |

**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** object.**New** [SaveChanges=**False**] [SaveFileName=""] [SetupOnly=**False**]

| Parameters  | Definition                                                                                    |
|-------------|-----------------------------------------------------------------------------------------------|
| object      | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.         |
| SaveChanges | A <b>Boolean</b> that specifies whether changes to the current configuration should be saved. |

| Parameters   | Definition                                                                                                                                                                                                                                     |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SaveFileName | A <b>String</b> that is the name of the file to which current configuration information should be saved.                                                                                                                                       |
| SetupOnly    | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

## 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]`

| Parameters   | Definition                                                                                                                                                                                                                                     |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object       | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                                                                          |
| OpenFileName | A <b>String</b> that contains the name of the file located on the Instrument.                                                                                                                                                                  |
| SaveChanges  | A <b>Boolean</b> that specifies whether changes to the current configuration should be saved.                                                                                                                                                  |
| SaveFileName | A <b>String</b> containing the name of the file to which current configuration information should be saved.                                                                                                                                    |
| SetupOnly    | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information. |

**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](#)) ]



**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** object.**PanelLock** [Message = ""]

| Parameters | Definition                                                                                                                                   |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                        |
| Message    | A <b>String</b> containing a custom message that will be shown on the display. If the message is empty, then a default message will be used. |

**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
// 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);

    // 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 = strlen(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);
}

```

## PanelUnlock Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 217](#)) ]

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Re-enables user access to the instrument front panel or remote display.

**VB Syntax** object.**PanelUnlock**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**See Also** • PanelLock (see [page 216](#)) method

## QueryCommand Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 175](#)) ]

**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 | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Query      | <p>A <b>String</b> that contains either an XML-based query command or an XML-based filter.</p> <p>If a filter is specified, the query command will be set to "GetAllSetup", 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.</p> <p>If the 'Query' starts with the character '&lt;', it is treated as an XML-based filter (see <a href="#">page 221</a>); otherwise, it is treated as an XML-based query (see <a href="#">page 221</a>) command.</p> |

| Returns    | Definition                                           |
|------------|------------------------------------------------------|
| XMLCommand | A <b>String</b> that contains the XML-based command. |

**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:

```
' 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:

```
myInst.QueryCommand "<Setup><Module><BusSignalSetup><BusSignal>",
_
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:

```
myInst.QueryCommand "<Setup><Module><BusSignalSetup>" + _
    "<BusSignal Name='My Bus 2' Polarity>", queryOutput
```

queryOutput contains:

```
Polarity="Positive"
```

- 5 White space is ignored. For example:

```
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

| Parameters      | Definition                                                                                                     |
|-----------------|----------------------------------------------------------------------------------------------------------------|
| object          | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object.                      |
| TriggerFileName | A <b>String</b> that contains the name of the XML-format trigger specification file located on the instrument. |

**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

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

**Applies To** • AnalyzerModule (see [page 106](#)) object

**Description** Loads a named trigger from the recall buffer.

**VB Syntax** object.**RecallTriggerByName** TriggerName

| Parameters  | Definition                                                                                |
|-------------|-------------------------------------------------------------------------------------------|
| object      | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object. |
| TriggerName | A <b>String</b> that represents the recall buffer title name.                             |

**See Also** • AnalyzerModule (see [page 106](#)) object

## RecvFile Method

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

**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

| Parameters   | Definition                                                                              |
|--------------|-----------------------------------------------------------------------------------------|
| object       | An expression that evaluates to a ConnectSystem (see <a href="#">page 119</a> ) object. |
| SrcFileName  | A <b>String</b> that is the name of the file on the remote logic analyzer system.       |
| DestFileName | A <b>String</b> that is the name of the file on the local file system.                  |

**See Also** • Connect (see [page 172](#)) method

## Remove Method (BusSignals Object)

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

**Applies To** • BusSignals (see [page 113](#)) object

**Description** Removes a bus/signal from the BusSignals collection.



**VB Syntax** BusSignals (see [page 113](#)).**Remove** Name

| Parameters | Definition                                                           |
|------------|----------------------------------------------------------------------|
| Name       | A <b>String</b> containing the name of the bus/signal to be removed. |

## 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

| Parameters | Definition                                                                               |
|------------|------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>Markers</b> (see <a href="#">page 123</a> ) object. |
| Name       | A <b>String</b> containing the name of the marker to be removed.                         |

## 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**

| Parameters | Definition                                                                               |
|------------|------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>Markers</b> (see <a href="#">page 123</a> ) object. |

## 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

| Parameters | Definition                                                                                                                          |
|------------|-------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>Markers</b> (see <a href="#">page 123</a> ) object.                                            |
| XMLMarkers | An "XML format" (in the online help) <b>String</b> containing the markers to remove (see "<Markers> element" (in the online help)). |

## RemoveLine Method

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Removes the instruction or vector at line number.

**VB Syntax** object.**RemoveLine** LineNumber

| Parameters | Definition                                                                                                                                                                                                                                                                                                                                      |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object.                                                                                                                                                                                                                                                         |
| LineNumber | A <b>String</b> representing the instruction/vector line number or line number range to be removed. For example: <ul style="list-style-type: none"> <li>• "5" — removes line 5.</li> <li>• "5..12" — removes lines 5 through 12.</li> <li>• "5:12" — another way to remove lines 5 through 12.</li> <li>• "All" — removes all lines.</li> </ul> |

**See Also** • InsertLine (see [page 214](#)) method

## Reset Method (PattgenModule Object)

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Resets the current line number to the first line.

**VB Syntax** object.**Reset**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |

**See Also** • Run (see [page 228](#)) method  
• Stop (see [page 236](#)) method

- Step (see [page 235](#)) method
- Resume (see [page 227](#)) method

## Resume Method (PattgenModule Object)

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Resumes running the pattern generator from the current line number.

**VB Syntax** object.**Resume**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |

- 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)

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

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Starts running all modules.

**VB Syntax** object.**Run** [Repetitive=**False**]

| Parameters | Definition                                                                                                                                                                                        |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                                             |
| Repetitive | A Boolean indicating the number of times the module(s) are started. <b>True</b> – run continuously until the Stop (see <a href="#">page 236</a> ) method is called. <b>False</b> – run only once. |

- See Also**
- Stop (see [page 236](#)) method
  - Status (see [page 280](#)) property
  - WaitComplete (see [page 238](#)) method

## Run Method (PattgenModule Object)

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Starts running the pattern generator.

**VB Syntax** object.**Run** [Repetitive=**False**]

| Parameters | Definition                                                                                                                                                                                       |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object.                                                                                                          |
| Repetitive | A Boolean indicating the number of times the module is started.<br><b>True</b> – run continuously until the Stop (see <a href="#">page 236</a> ) method is called. <b>False</b> – run only once. |

**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

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

**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**]

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

| Parameters   | Definition                                                                                                                                                                                                                                                                                                     |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SaveFileName | A <b>String</b> that contains the name of the file to be saved on the Instrument.<br>If the filename string has a suffix of ".xml", the configuration is saved to an XML format file; if the filename string has a suffix of ".ala" or has no suffix at all, the configuration is saved to an ALA format file. |
| SetupOnly    | A <b>Boolean</b> that specifies whether the file to which the current configuration will be saved contains captured data or just setup information: <b>True</b> – save only setup information. <b>False</b> – save data and setup information.                                                                 |

**Remarks** The file is stored onto a drive that is directly accessible by the instrument.

**See Also** • Open (see [page 216](#)) method

## SendFile Method

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

**Applies To** • **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

| Parameters   | Definition                                                                              |
|--------------|-----------------------------------------------------------------------------------------|
| object       | An expression that evaluates to a ConnectSystem (see <a href="#">page 119</a> ) object. |
| SrcFileName  | A <b>String</b> that is the name of the file on the local file system.                  |
| DestFileName | A <b>String</b> that is the name of the file on the remote logic analyzer system.       |

**See Also** • Connect (see [page 172](#)) method

## SetGroupCaption Method

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

**Applies To** • **VbaViewChartData** (see [page 157](#)) object

**Description** Sets the caption associated with a group (row).

**VB Syntax** object.**SetGroupCaption** Group, Caption

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Group      | A <b>Long</b> representing the group (row) on which to set the caption.                    |
| Caption    | A <b>String</b> that is the caption to be set.                                             |

**See Also** • [GetGroupCaption](#) (see [page 193](#)) method

## SetLine Method

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Sets an instruction or vector at line number.

**VB Syntax** object.**SetLine** LineNumber, InstructionOrVector

| Parameters          | Definition                                                                                           |
|---------------------|------------------------------------------------------------------------------------------------------|
| object              | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object.              |
| LineNumber          | A <b>Long</b> representing the line number at which the instruction or vector should be set.         |
| InstructionOrVector | A <b>String</b> containing the instruction or vector (see <a href="#">page 194</a> ) to be inserted. |

**See Also** • [GetLine](#) (see [page 194](#)) method

## SetLineLabel Method

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Sets a vector's label value at line number.

**VB Syntax** object.**SetLineLabel** LineNumber, LabelName, Value

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |
| LineNumber | A <b>Long</b> representing the line number from which to get the instruction or vector. |
| LabelName  | A <b>String</b> name of label whose value you wish to set.                              |
| Value      | A <b>String</b> representing the label's value.                                         |

**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

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Group      | A <b>Long</b> representing the group (row) on which to set the value.                      |
| Index      | A <b>Long</b> representing the index (column) on which to set the value.                   |
| Value      | A <b>Variant</b> that is the value to set in the chart.                                    |

**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

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Group      | A <b>Long</b> representing the group (row) on which to set the value array.                |
| ValueArray | An array of <b>Variants</b> that are the values to set in the chart.                       |

**See Also** • SetValue (see [page 231](#)) method

### SetValueCaption Method

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

**Applies To** • VbaViewChartData (see [page 157](#)) object

**Description** Sets the caption associated with all values at index (column).

**VB Syntax** object.SetValueCaption Index, Caption

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartData (see <a href="#">page 157</a> ) object. |
| Index      | A <b>Long</b> representing the index (column) on which to set the caption.                 |
| Caption    | A <b>String</b> that is the caption to be set.                                             |

**See Also** • GetValueCaption (see [page 206](#)) method

### SimpleTrigger Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 51](#)) ]

**Applies To** • AnalyzerModule (see [page 106](#)) object

**Description** Trigger on a simple condition with optional occurrence and storage qualification. Default triggers on anything and stores everything.

**VB Syntax** object.SimpleTrigger [OnEvent = "Anything"], [Occurs=1], [StoreEvent="Anything"]



| Parameters | Definition                                                                                                                    |
|------------|-------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object.                                     |
| OnEvent    | A <b>String</b> containing the Event (see <a href="#">page 233</a> ) to trigger on.                                           |
| Occurs     | A <b>Long</b> containing the number of occurrences of <b>OnEvent</b> before triggering.                                       |
| StoreEvent | A <b>String</b> containing the storage qualification Event (see <a href="#">page 233</a> ) while waiting for <b>OnEvent</b> . |

**Remarks** If the analyzer is in timing mode, the only valid value for **StoreEvent** is "Anything".

**See Also**

- RecallTriggerByFile (see [page 223](#)) method
- RecallTriggerByName (see [page 224](#)) method

### Simple Trigger/Find Event

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 51](#)) ]

**Applies To**

- SimpleTrigger** (see [page 232](#)) method
- Find** (see [page 184](#)) method

**Description** 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:**

|                   |                                                             |
|-------------------|-------------------------------------------------------------|
| hFFXX0022         | Hex number with 2 don't care digits (8 don't care bits).    |
| o7777xxxx         | Octal number with 4 don't care digits (12 don't care bits). |
| b10110110xxxx0000 | Binary number with 4 don't care bits.                       |

**Range examples:**

|              |                              |
|--------------|------------------------------|
| hff00..hffff | Range from hex ff00 to ffff. |
|--------------|------------------------------|

**Edge examples:**

|           |                                                                                           |
|-----------|-------------------------------------------------------------------------------------------|
| eXXXXRFEG | Edge specification with 4 don't care bits, then rising, falling, either, and glitch bits. |
| eR        | A rising edge on a signal (one bit).                                                      |

**Event examples:**

|                                               |                                                                                                   |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------|
| ADDR=hffffxxxx                                | Specifies a simple bus/signal value.                                                              |
| ADDR=hffff0000..hfffffff                      | Specifies the same simple bus/signal value as a range.                                            |
| ADDR=hffff0000..hfffffff<br>And DATA=eXXXXXXX | Specifies a rising edge in bit 5 of DATA while ADDR is within the hex range ffff0000 to ffffffff. |
| ADDR=hffxx And DATA=h055                      | Specifies AND'ing two bus/signal values.                                                          |

**Trigger Event example:**

|          |                      |
|----------|----------------------|
| Anything | Specifies any value. |
|----------|----------------------|

**Find Event example:**

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| ADDR entering hff00 | Specifies the first sample of one or more consecutive samples whose ADDR is hFF00. |
|---------------------|------------------------------------------------------------------------------------|

- See Also**
- SimpleTrigger (see [page 232](#)) method
  - Find (see [page 184](#)) method

## Step Method

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

**Applies To** • PattgenModule (see [page 129](#)) object

**Description** Steps the pattern generator from the current line number.

**VB Syntax** object.**Step** [Count=1]

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |
| Count      | A <b>Long</b> indicating the number of vectors the pattern generator should output.     |

- See Also**
- Run (see [page 228](#)) method
  - Stop (see [page 236](#)) method

## Stop Method (Instrument Object)

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

**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**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**See Also** • Run (see [page 227](#)) method  
• Status (see [page 280](#)) property

## Stop Method (PattgenModule Object)

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

**Applies To** • **PattgenModule** (see [page 129](#)) object

**Description** Stops the pattern generator if it is currently running.

**VB Syntax** object.**Stop**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |

**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**

| Parameters | Definition                                                                         |
|------------|------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a SelfTest (see <a href="#">page 149</a> ) object. |

**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"

## 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  | Definition                                                                                                                                                                          |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object      | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.                                                                                               |
| ProjectName | A <b>String</b> 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. |
| HelpTopic   | A <b>String</b> that is the name of a topic in the help file.                                                                                                                       |

## 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** `object.VBARunMacro MacroName`

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |
| MacroName  | A <b>String</b> that is the name of the VBA macro to run.                             |

## 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** `object.VBARunRPICommand Command`

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |
| Command    | A <b>String</b> containing the ASCII RPI command to run.                              |

**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**    `object.WaitComplete [Seconds="- 1"]`

| Parameters | Definition                                                                                                                                                                                                                                                           |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above.                                                                                                                                                                                   |
| Seconds    | A <b>Long</b> containing the maximum number of seconds to wait for the measurement to complete.<br><b>Note:</b> If set to -1, this method does not return until the specified measurement completes; therefore, we strongly recommend you make this value $\geq 0$ . |

### WaitComplete Example

**Visual Basic**

```

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."
    End Select
End Sub

```

```

        Resume Next
    Case Else
        Err.Raise Number:=Err.Number
    End Select

End Sub

```

```

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::IIInstrumentPtr 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::IIAnalyzerModulePtr 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 = strlen(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** object.**WriteOutput** String

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWindow (see <a href="#">page 160</a> ) object. |
| String     | A <b>String</b> to be written to the VbaView window.                                    |

## 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**

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>BusSignal</b> (see <a href="#">page 108</a> ) object. |

**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

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

**Applies To** • **VbaViewChart** (see [page 156](#)) object

**Description** Gets the chart axis given an axis type.

**VB Syntax** object.**Axis** AxisType

| Parameters | Definition                                                                                                   |
|------------|--------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChart (see <a href="#">page 156</a> ) object.                       |
| AxisType   | An <b>AgtChartAxisType</b> enumerated type value that identifies the X or Y axis. See the description below. |

**Remarks** The **Axis** property has the **VbaViewChartAxis** (see [page 157](#)) object type. The **AxisType** parameter can have the following values:

| AgtChartAxisType  | Enum Value | Description |
|-------------------|------------|-------------|
| AgtChartAxisTypeX | 1          |             |
| AgtChartAxisTypeY | 2          |             |

## AxisBase Property

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

**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 | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartAxis (see <a href="#">page 157</a> ) object. |
| Base       | An <b>AgtAxisBase</b> enumerated type value that can be one of the values described below. |

**Remarks** The **AxisBase** property can have the following values:

| AgtAxisBase              | Enum Value | Description                      |
|--------------------------|------------|----------------------------------|
| AgtAxisBaseBinary        | 1          |                                  |
| AgtAxisBaseHex           | 2          |                                  |
| AgtAxisBaseOctal         | 3          |                                  |
| AgtAxisBaseDecimal       | 4          |                                  |
| AgtAxisBaseSignedDecimal | 5          |                                  |
| AgtAxisBaseAscii         | 6          |                                  |
| AgtAxisBaseFloatingPoint | 7          | BitSize is ignored for this type |

**See Also** • BitSize (see [page 248](#)) property (of VbaViewChartAxis object)

## 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                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Marker (see <a href="#">page 123</a> ) object. |
| Color      | A <b>Long</b> value that is the marker background color.                         |

**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

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

**Applies To** • **BusSignal** (see [page 108](#)) object

**Description** Gets the number of channels in the bus/signal.

**VB Syntax** object.**BitSize**

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an <b>BusSignal</b> (see <a href="#">page 108</a> ) object. |

**Remarks** The **BitSize** property has the **Long** type.

**See Also** • **ByteSize** (see [page 251](#)) property

## BitSize Property (of VbaViewChartAxis)

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

**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]

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an VbaViewChartAxis (see <a href="#">page 157</a> ) object. |
| BitSize    | A <b>Long</b> value that specifies the width of the data in bits.                           |

**Remarks** The **BitSize** property has the **Long** type.

**See Also** • **AxisBase** (see [page 246](#)) property

## Bold 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 thickness.

**VB Syntax** object.**Bold** [=Bold]



| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartFont (see <a href="#">page 158</a> ) object. |
| Bold       | A <b>Boolean</b> that specifies whether the text is bold.                                  |

**Remarks** The **Bold** property has the **Boolean** type.

## BusSignalData Property

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

**Applies To** • **BusSignal** (see [page 108](#)) object

**Description** Gets the data associated with a bus/signal.

**VB Syntax** object.**BusSignalData**

| Parameters | Definition                                                                          |
|------------|-------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object. |

**Remarks** The **BusSignalData** property has the **BusSignalData** (see [page 108](#)) object type.

## BusSignalType Property

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

**Applies To** • **BusSignal** (see [page 108](#)) object

**Description** Gets the type of bus/signal.

**VB Syntax** object.**BusSignalType**

| Parameters | Definition                                                                          |
|------------|-------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object. |

**Remarks** The **BusSignalType** property can have the following values:

| AgtBusSignalType   | Enum Value | Description                                                       |
|--------------------|------------|-------------------------------------------------------------------|
| AgtBusSignalProbed | 1          | The bus/signal is associated with a physically probed connection. |

|                       |   |                                                                                                                                                                                                                                                                                            |
|-----------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AgtBusSignalGenerated | 2 | The bus/signal is generated by a tool (like an inverse assembler) and therefore does not have a physically probed connection. This type is also used for an external oscilloscope because its bus/signals are not physically probed directly by the logic analyzer.                        |
| AgtBusSignalSampleNum | 3 | Represents the sample number.                                                                                                                                                                                                                                                              |
| AgtBusSignalTime      | 4 | Represents the sample's absolute time. This is obsolete. The "Time" data is no longer returned as a bus/signal. To get the time associated with bus/signal data, use the GetTime (see <a href="#">page 204</a> ) method of the SampleBusSignalData (see <a href="#">page 138</a> ) object. |

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](#)) ]

**Applies To** • **SampleDifference** (see [page 149](#)) object

**Description** Gets a collection of all the buses/signals with differences for this sample.

**VB Syntax** object.**BusSignalDifferences**

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a SampleDifference (see <a href="#">page 149</a> ) object. |

**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](#)) ]

**Applies 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**

| Parameters | Definition                                                                        |
|------------|-----------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the <b>Applies to</b> line. |

**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**

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an BusSignal (see <a href="#">page 108</a> ) object. |

**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]

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartTitle (see <a href="#">page 159</a> ) object. |
| Caption    | A <b>String</b> that represents the chart title caption.                                    |

**Remarks** The **Caption** property has the **String** type.

## CardModels Property

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

**Applies To** • **Module** (see [page 128](#)) object

**Description** Gets the card model numbers.

**VB Syntax** object.**CardModels**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Module (see <a href="#">page 128</a> ) object. |

**Remarks** The **CardModels** property has the **String** type.

## Channels Property

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

**Applies To** • **BusSignal** (see [page 108](#)) object

**Description** Gets the channels defined in the bus/signal.

**VB Syntax** object.**Channels**

| Parameters | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Channels   | <p>A <b>String</b> containing <i>MultiplePodChannels</i> or the <b>String</b> "None" if no channels are assigned.</p> <p><b>MultiplePodChannels</b> contains a comma separated list of <i>PodChannels</i>.</p> <p><b>PodChannels</b> contains a <i>PodNumber</i> followed by the <b>String</b> "[" followed by a comma separated list of individual channel <i>Number(s)</i> and <i>NumberRange(s)</i> followed by the <b>String</b> "]".</p> <p><b>Note:</b> Pod channels normally are in MSB to LSB notation unless you are trying to reorder channels.</p> <p><b>NumberRange</b> contains a <i>Number</i> followed by the <b>String</b> ":" followed by a <i>Number</i>.</p> <p>PodChannels Example:<br/> Pod 1 [9 : 7 , 5 , 3 : 1] — this bus consists of 1 single channel <i>Number</i> and 2 <i>NumberRange</i>'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].</p> <p>MultiplePodChannels Example:<br/> Pod 2 [3 , 1] , Pod 3 [10 , 5 : 3] — this bus consists of 3 single channel <i>Numbers</i> and 1 <i>NumberRange</i> for a total of 6 channels Pod 2[3], Pod 2[1], Pod 3[10], Pod 3[5], Pod 3[4], Pod 3[3].</p> |

**Remarks** The **Channels** property has the **String** type.

## Chart Property

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

**Applies To** • **VbaViewWindow** (see [page 160](#)) object

**Description** Gets the Chart view.

**VB Syntax** object.**Chart**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWindow (see <a href="#">page 160</a> ) object. |

**Remarks** The **Chart** property has the **VbaViewChart** (see [page 156](#)) object type.

## ChartType 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 the chart type.

**VB Syntax** object.**ChartType** [=Type]

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChart (see <a href="#">page 156</a> ) object.      |
| Type       | An <b>AgtChartType</b> enumerated type value that can be one of the values described below. |

**Remarks** The **ChartType** property can have the following values:

| AgtChartType                     | Enum Value | Description |
|----------------------------------|------------|-------------|
| AgtChartTypeNone                 | 1          |             |
| AgtChartTypeLine                 | 2          |             |
| AgtChartTypeLineOnly             | 3          |             |
| AgtChartTypeXYScatter            | 4          |             |
| AgtChartTypeHorizontalBar        | 5          |             |
| AgtChartTypeVerticalBar          | 6          |             |
| AgtChartTypePie                  | 7          |             |
| AgtChartTypeStackedVerticalBar   | 8          |             |
| AgtChartTypeStackedHorizontalBar | 9          |             |

## 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]

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartFont (see <a href="#">page 158</a> ) object. |
| Color      | A <b>Long</b> value that represents the text color.                                        |

**Remarks** 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

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

**Applies To** • **Marker** (see [page 123](#)) object

**Description** Gets or sets the marker comments.

**VB Syntax** object.**Comments** [=Comments]

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Marker (see <a href="#">page 123</a> ) object. |
| Comments   | A <b>String</b> containing the comments for the marker.                          |

**Remarks** The **Comments** property has the **String** type.

## ComputerName Property

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

**Applies To** • **Frame** (see [page 120](#)) object

**Description** Gets a frame's computer name.

**VB Syntax** object.**ComputerName**

- or -

object

| Parameters | Definition                                                                                                                                                                                                                                                                                                    |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object     | An expression that evaluates to a <b>Frame</b> (see <a href="#">page 120</a> ) object. The <b>ComputerName</b> property is the default property of the <b>Frame</b> (see <a href="#">page 120</a> ) object. Accordingly, you do not have to reference <b>ComputerName</b> explicitly, as shown in the syntax. |

**Remarks** The **ComputerName** property has the **String** type.

## Count Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 256](#)) ]

- 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 the number of items in a collection.

**VB Syntax** object.Count

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the <b>Applies to</b> list above. |

**Remarks** The **Count** property has the **Long** type.

**See Also**

- **Item** (see [page 266](#)) property

## Item and Count Example

The following example displays the channels for each bus/signal in the **BusSignals** collection:

**Visual Basic**

```
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)
```



```

        ' Add the bus/signal name and channels to the string.
        myBusSignalChannels = myBusSignalChannels + vbNewLine
        myBusSignalChannels = myBusSignalChannels + "Bus/signal: " + _
            myBusSignal.Name
        myBusSignalChannels = myBusSignalChannels + ", Channels: " + _
            myBusSignal.Channels
    Next
    MsgBox "Bus/signal names and channels: " + vbNewLine + _
        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::IIInstrumentPtr 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 = strlen(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

| Parameters | Definition                                                                          |
|------------|-------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object. |

**Remarks** The **CreatorName** property has the **String** type.

## Data Property

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

**Applies To** • **VbaViewChart** (see [page 156](#)) object

**Description** Gets the chart axis given an axis type.

**VB Syntax** object.**Data**

| Parameters | Definition                                                                             |
|------------|----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChart (see <a href="#">page 156</a> ) object. |

**Remarks** The **Data** property has the **VbaViewChartData** (see [page 157](#)) object type.

## DataType Property

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

**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**

| Parameters | Definition                                                                                    |
|------------|-----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a SampleBusSignalData (see <a href="#">page 138</a> ) object. |

**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

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

**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**

| Parameters | Definition                                                                                                                |
|------------|---------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Frame (see <a href="#">page 120</a> ) or Module (see <a href="#">page 128</a> ) object. |

**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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a CompareWindow (see <a href="#">page 118</a> ) object. |

**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**

| Parameters | Definition                                                                                     |
|------------|------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object. |

**Remarks** The **EndSample** property has the **Long** type.

**See Also** • StartSample (see [page 279](#)) property

## EndTime Property

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

**Applies To** • SampleBusSignalData (see [page 138](#)) object

**Description** Gets the data's ending time (in seconds) relative to trigger.

**VB Syntax** object.**EndTime**

| Parameters | Definition                                                                                     |
|------------|------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object. |

**Remarks** The **EndTime** property has the **Double** type.

**See Also** • StartTime (see [page 279](#)) property

## FaceName 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 face name string.

**VB Syntax** object.**FaceName** [=FaceName]

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartFont (see <a href="#">page 158</a> ) object. |
| FaceName   | A <b>String</b> value that is the font typeface name.                                      |

**Remarks** The **FaceName** property has the **String** type.

## Font Property

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

**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 VbaViewChartTitle (see <a href="#">page 159</a> ) 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 <a href="#">page 119</a> ) 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 <a href="#">page 128</a> ) 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**

| Parameters | Definition                                                                                   |
|------------|----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an <b>Instrument</b> (see <a href="#">page 121</a> ) object. |

**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]

| Parameters | Definition                                                                                    |
|------------|-----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>VbaViewChart</b> (see <a href="#">page 156</a> ) object. |
| HasLegend  | A <b>Boolean</b> value that specifies whether the legend is visible.                          |

**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]



| Parameters | Definition                                                                         |
|------------|------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above. |
| HasTitle   | A <b>Boolean</b> value that specifies whether the title is visible.                |

**Remarks** The **HasTitle** property has the **Boolean** type.

## Instrument Property

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

**Applies To** • **Connect** (see [page 118](#)) object

**Description** Gets the logic analyzer instrument object.

**VB Syntax** object.**Instrument** [HostNameOrIpAddress=""]

| Parameters          | Definition                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object              | An expression that evaluates to a Connect (see <a href="#">page 118</a> ) object.                                                                                                                                                                                                                                                                                                          |
| HostNameOrIpAddress | A <b>String</b> that contains the hostname or IP address of the logic analyzer instrument or computer on which the <i>Agilent Logic Analyzer</i> application runs. This parameter is optional—if it is not specified, the computer name specified in the Distributed COM "Location" tab is used (see To verify remote computer Distributed COM properties (see <a href="#">page 39</a> )). |

**Remarks** The **Instrument** property has the **Instrument** (see [page 121](#)) object type.

## IPAddress Property

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

**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** object.**IPAddress**

| Parameters | Definition                                                                             |
|------------|----------------------------------------------------------------------------------------|
| Object     | An expression that evaluates to a <b>Frame</b> (see <a href="#">page 120</a> ) object. |

**Remarks** The **IPAddress** property has the **String** type.

## Item Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 256](#)) ]

- 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** `object.Item [IndexOrName]`

- or -

`object IndexOrName`

| Parameters  | Definition                                                                                                                         |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| Object      | An expression that evaluates to one of the objects in the Applies To list above.                                                   |
| IndexOrName | A <b>Variant</b> that is a <b>Long</b> or <b>String</b> representing the zero-based index in the collection or the name of object. |

| Object                                               | String Index                                                                                                                                                                    |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BusSignalDifferences (see <a href="#">page 109</a> ) | The BusSignalDifference's index.                                                                                                                                                |
| BusSignals (see <a href="#">page 113</a> )           | The BusSignal's name (as defined by the Name (see <a href="#">page 269</a> ) property).                                                                                         |
| Frames (see <a href="#">page 120</a> )               | The Frame's computer name or IP address (as defined by the ComputerName (see <a href="#">page 255</a> ) or IPAddress (see <a href="#">page 265</a> ) properties, respectively). |
| Markers (see <a href="#">page 123</a> )              | The Marker's name (as defined by the Name (see <a href="#">page 269</a> ) property).                                                                                            |
| Modules (see <a href="#">page 129</a> )              | The Module's name (as defined by the Name (see <a href="#">page 269</a> ) property).                                                                                            |

| Object                                            | String Index                                                                         |
|---------------------------------------------------|--------------------------------------------------------------------------------------|
| Probes (see <a href="#">page 135</a> )            | The Probe's name (as defined by the Name (see <a href="#">page 269</a> ) property).  |
| SampleDifferences (see <a href="#">page 149</a> ) | The SampleDifference's index.                                                        |
| Tools (see <a href="#">page 153</a> )             | The Tool's name (as defined by the Name (see <a href="#">page 269</a> ) property).   |
| Windows (see <a href="#">page 161</a> )           | The Window's name (as defined by the Name (see <a href="#">page 269</a> ) property). |

**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 | Definition                                                                                    |
|------------|-----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a <b>VbaViewChart</b> (see <a href="#">page 156</a> ) object. |

**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**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Module (see <a href="#">page 128</a> ) object. |

**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:

| Object                                     | Results                                                                                                                                                                                                                                                                          |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Instrument (see <a href="#">page 121</a> ) | Gets the instrument's model number.                                                                                                                                                                                                                                              |
| Module (see <a href="#">page 128</a> )     | Gets the module's model number. If the module consists of different card model numbers, they are separated with spaces. For example, an analyzer model would look like "16756A"; a two-card analyzer module containing two different card types would look like "16750A 16750B". |

## 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**

| Parameters | Definition                                                                                   |
|------------|----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an <b>Instrument</b> (see <a href="#">page 121</a> ) object. |

**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

| Parameters | Definition                                                                                                                                                                                                                                |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object     | 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. |
| NewName    | A <b>String</b> containing the new name of the object.                                                                                                                                                                                    |

**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:

| Object                                              | Results                                                                                              |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------|
| BusSignal (see <a href="#">page 108</a> )           | Gets or sets the name of the bus/signal.                                                             |
| BusSignalDifference (see <a href="#">page 109</a> ) | Gets the bus/signal name associated with the bus/signal difference.                                  |
| Marker (see <a href="#">page 123</a> )              | Gets or sets the name of the marker.                                                                 |
| Module (see <a href="#">page 128</a> )              | Gets or sets the name of the module. This is the name displayed in the instrument's Overview window. |
| Probe (see <a href="#">page 135</a> )               | Gets or sets the name of the probe. This is the name displayed in the instrument's Overview window.  |
| Tool (see <a href="#">page 152</a> )                | Gets or sets the name of the tool. This is the name displayed in the instrument's Overview window.   |
| Window (see <a href="#">page 160</a> )              | Gets or sets the name of the window. This is the name displayed in the instrument's Overview window. |

## 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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a PattgenModule (see <a href="#">page 129</a> ) object. |

**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**

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a FindResult (see <a href="#">page 119</a> ) object. |

**Remarks** The **OccurrencesFound** property has the **Long** type.

## Options Property

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

**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** object.**Options** [=Options]

| Parameters | Definition                                                                                                            |
|------------|-----------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a CompareWindow (see <a href="#">page 118</a> ) object.                               |
| Options    | A <b>String</b> containing the "XML format" (in the online help)"<Options> element" (in the online help) information. |

**Remarks** The **Options** property has the **String** type.

## Overview Property

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

**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.
Dim myOverview As String
myOverview = myInst.Overview
MsgBox myOverview
```

**VB Syntax** object.**Overview**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object.       |
| Message    | If the front panel is locked, a <b>String</b> containing the displayed message is returned. |

**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**

| Parameters | Definition                                                                          |
|------------|-------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object. |

**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**    **object.Position** [=Time]

| Parameters | Definition                                                                                     |
|------------|------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Marker (see <a href="#">page 123</a> ) object.               |
| Time       | A <b>Double</b> value that is the marker's time position, in seconds, relative to the trigger. |

**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**    Gets or sets the chart legend position.

**VB Syntax**    **object.Position** [=Position]

| Parameters | Definition                                                                                       |
|------------|--------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartLegend (see <a href="#">page 158</a> ) object.     |
| Position   | An <b>AgtLegendPosition</b> enumerated type value that can be one of the values described below. |

**Remarks**    The **LegendPosition** property can have the following values:

| AgtLegendPosition       | Enum Value | Description |
|-------------------------|------------|-------------|
| AgtLegendPositionBottom | 1          |             |
| AgtLegendPositionLeft   | 2          |             |
| AgtLegendPositionRight  | 3          |             |
| AgtLegendPositionTop    | 4          |             |

## Probes Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 105](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Gets a collection of all currently defined probes.

**VB Syntax** object.Probes

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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

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

**Applies To** • **BusSignalDifference** (see [page 109](#)) object

**Description** Gets the reference buffer value associated with the bus/signal difference.

**VB Syntax** object.Reference

| Parameters | Definition                                                                                    |
|------------|-----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignalDifference (see <a href="#">page 109</a> ) object. |

**Remarks** The **Reference** property has the **String** type.

The value is in base hex, for example: "FF".

## RemoteComputerName Property

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

**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]

| Parameters         | Definition                                                                           |
|--------------------|--------------------------------------------------------------------------------------|
| object             | An expression that evaluates to a Instrument (see <a href="#">page 121</a> ) object. |
| RemoteComputerName | A <b>String</b> value that is the name of the remote computer.                       |

**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]

| Parameters     | Definition                                                                           |
|----------------|--------------------------------------------------------------------------------------|
| object         | An expression that evaluates to a Instrument (see <a href="#">page 121</a> ) object. |
| RemoteUserName | A <b>String</b> value that is the login name of the remote user.                     |

**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**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Module (see <a href="#">page 128</a> ) object. |

**Remarks** The **RunningStatus** property has the **String** type. The string returned is based on the object types defined below.

| Object                                 | Description                                                                                                                                                                                                                                    |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Module (see <a href="#">page 128</a> ) | Running - the module is running<br>Stopped - the module has stopped<br>running<br>Initializing - the module is initializing or calibrating<br>Waiting - the module is waiting for an event<br>SelfTest - the instrument is running<br>SelfTest |

**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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a CompareWindow (see <a href="#">page 118</a> ) object. |

**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** object.**SampleNum**

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a SampleDifferences (see <a href="#">page 149</a> ) object. |

**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** object.**SelfTest**

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Instrument (see <a href="#">page 121</a> ) object. |

**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.

```
' 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**    object.**Setup** [=XMLSetupSpec]

| Parameters   | Definition                                                                                                           |
|--------------|----------------------------------------------------------------------------------------------------------------------|
| object       | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object.                            |
| XMLSetupSpec | A <b>String</b> containing the "XML format" (in the online help)"<Module> element" (in the online help) information. |

**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**    object.**Size** [=Size]

| Parameters | Definition                                                                                 |
|------------|--------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewChartFont (see <a href="#">page 158</a> ) object. |
| Size       | A <b>Long</b> value that specifies the text size in points.                                |

**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**    object.**Slot**

| Parameters | Definition                                                                       |
|------------|----------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Module (see <a href="#">page 128</a> ) object. |

**Remarks** The **Slot** property has the **String** type.

**See Also**

- **Frame** (see [page 120](#)) object
- **Frames** (see [page 120](#)) object

## StartSample Property

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

**Applies To**

- **SampleBusSignalData** (see [page 138](#)) object

**Description** Gets the data's starting sample number relative to trigger.

**VB Syntax** object.**StartSample**

| Parameters | Definition                                                                                     |
|------------|------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object. |

**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

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

**Applies To**

- **SampleBusSignalData** (see [page 138](#)) object

**Description** Gets the data's starting time (in seconds) relative to trigger.

**VB Syntax** object.**StartTime**

| Parameters | Definition                                                                                     |
|------------|------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an SampleBusSignalData (see <a href="#">page 138</a> ) object. |

**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**

| Parameters | Definition                                                                         |
|------------|------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above. |

**Remarks** The **Status** property has the **String** type. The string returned is based on the object types defined below.

| Object                                     | Description                                                                                                                                                        |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Instrument (see <a href="#">page 121</a> ) | Running - at least one of the data acquisition modules, tools, or windows is running<br>Stopped - data acquisition modules, tools, or windows have stopped running |
| Module (see <a href="#">page 128</a> )     | Running - the module is running<br>Stopped - the module has stopped running                                                                                        |

**See Also** • RunningStatus (see [page 275](#)) property



## StatusMsg Property

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

**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 <a href="#">page 128</a> ) 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

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

**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 <a href="#">page 119</a> ) object. |

**Remarks** The **SubrowFound** property has the **Long** type.

## Symbols Property

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

**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]

| Parameters | Definition                                                                                                            |
|------------|-----------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignal (see <a href="#">page 108</a> ) object.                                   |
| XMLSymbols | A <b>String</b> containing the "XML format" (in the online help)"<Symbols> element" (in the online help) information. |

**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]

| Parameters | Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a Frame (see <a href="#">page 120</a> ) object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Value      | <p>A <b>String</b> 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:</p> <ul style="list-style-type: none"> <li>• h - for hexadecimal</li> <li>• b - binary</li> <li>• o - octal</li> <li>• d - decimal</li> </ul> <p>For example: "hf", "b11110000", "bxxxx1xxx".</p> |

**Remarks** The **TargetControlPort** property has the **String** type.

## TextColor Property

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

**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 <a href="#">page 123</a> ) object. |
| Color      | A <b>Long</b> 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

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

**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 <a href="#">page 119</a> ) 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** object.**TimeFoundString**

| Parameters | Definition                                                                           |
|------------|--------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a FindResult (see <a href="#">page 119</a> ) object. |

**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** object.**Title**

| Parameters | Definition                                                                         |
|------------|------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above. |

**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** object.**Tools**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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.

```
' 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** object.**Trigger** [=XMLTriggerSpec]

| Parameters     | Definition                                                                                                            |
|----------------|-----------------------------------------------------------------------------------------------------------------------|
| object         | An expression that evaluates to an AnalyzerModule (see <a href="#">page 106</a> ) object.                             |
| XMLTriggerSpec | A <b>String</b> containing the "XML format" (in the online help)"<Trigger> element" (in the online help) information. |

**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

| Parameters | Definition                                                                         |
|------------|------------------------------------------------------------------------------------|
| object     | An expression that evaluates to one of the objects in the "Applies to" list above. |

**Remarks** The **Type** property has the **String** type.

The following tables show how types correspond to objects.

| BusSignalData Type | BusSignalData Object                                |
|--------------------|-----------------------------------------------------|
| Sample             | SampleBusSignalData (see <a href="#">page 138</a> ) |

| Module Type   | Module Object                                  |
|---------------|------------------------------------------------|
| Analyzer      | AnalyzerModule (see <a href="#">page 106</a> ) |
| ExternalScope | Module (see <a href="#">page 128</a> )         |
| Import        | Module (see <a href="#">page 128</a> )         |
| Pattgen       | PattgenModule (see <a href="#">page 129</a> )  |

| Window Type | Window Object                                 |
|-------------|-----------------------------------------------|
| Compare     | CompareWindow (see <a href="#">page 118</a> ) |

## Value Property

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

**Applies To** • **BusSignalDifference** (see [page 109](#)) object

**Description** Gets the data value associated with the bus/signal difference.

**VB Syntax** object.Value

| Parameters | Definition                                                                                    |
|------------|-----------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a BusSignalDifference (see <a href="#">page 109</a> ) object. |

**Remarks** The **Value** property has the **String** type.  
The value is in base hex, for example: "FF".

## VBAVersion Property

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

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Gets the version number of VBA.

**VB Syntax** object.VBAVersion

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**Remarks** The **VBAVersion** property has the **String** type.

## VBE Property

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

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Gets the VBE extensibility object.

**VB Syntax** object.VBE

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**Remarks** The **VBE** property has the **VBE** object type.

## Version Property

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

**Applies To** • **Instrument** (see [page 121](#)) object

**Description** Gets the version number of the system software.

**VB Syntax** object.**Version**

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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**

| Parameters | Definition                                                                              |
|------------|-----------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWindow (see <a href="#">page 160</a> ) object. |

**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**

| Parameters | Definition                                                                                  |
|------------|---------------------------------------------------------------------------------------------|
| object     | An expression that evaluates to a VbaViewWebBrowser (see <a href="#">page 159</a> ) object. |



**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`

| Parameters | Definition                                                                            |
|------------|---------------------------------------------------------------------------------------|
| object     | An expression that evaluates to an Instrument (see <a href="#">page 121</a> ) object. |

**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`

| Parameters | Definition                                                                                                                                                                                                                                                                                                                          |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object     | With Visual C++, you can browse a collection to find a particular item by using the <b>_NewEnum</b> property or the <b>Item</b> (see <a href="#">page 266</a> ) property. In Visual Basic, you do not need to use the <b>_NewEnum</b> property because it is automatically used in the implementation of <b>For Each ... Next</b> . |

**Remarks** 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's their names.

#### **Visual Basic**

```
' 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 + vbNewLine + myBusSignal.Name
Next
MsgBox "Bus/signal names: " + myBusSignalNames
```

#### **Visual C++**

```
//
// 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/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::IIInstrumentPtr 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());

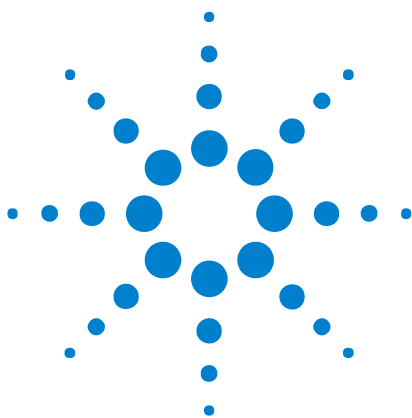
    static char errorStr[1024];
    _bstr_t desc = error.Description();

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

\_NewEnum example, [290](#)  
\_NewEnum property, [289](#)

## Numerics

16900 service, [20](#)

## A

Activity property, [245](#)  
Add method, BusSignals object, [168](#)  
Add method, Markers object, [169](#)  
AddPointArrays method, [170](#)  
AddXML method, [170](#)  
advanced triggers, COM automation, [54](#)  
agClientSvr.tlh header file, Visual C++, [70](#)  
agFirewSP2.wsf, [18](#)  
Agilent Logic Analysis Application program, [18](#)  
Agilent Logic Analysis Service program, [18](#)  
AgtDataType, [190](#)  
AnalyzerModule object, [106](#)  
automation overview, [13](#)  
Axis property, [246](#)  
AxisBase property, [246](#)

## B

BackgroundColor property, [247](#)  
BitSize property, [248](#)  
BitSize property (of VbaViewChartAxis object), [248](#)  
Bold property, [248](#)  
BusSignal object, [108](#)  
BusSignalData object, [108](#)  
BusSignalData property, [249](#)  
BusSignalDifference object, [109](#)  
BusSignalDifferences example, [109](#)  
BusSignalDifferences object, [109](#)  
BusSignalDifferences property, [250](#)  
BusSignals example, [114](#)  
BusSignals object, [113](#)  
BusSignals property, [250](#)  
BusSignalType property, [249](#)  
ByteSize property, [251](#)

## C

Caption property, [251](#)  
CardModels property, [252](#)  
changes since previous releases, [293](#)

Channels property, [252](#)  
Chart property, [253](#)  
ChartType property, [253](#)  
Clear method, [171](#)  
ClearOutput method, [172](#)  
Close method, [172](#)  
Color property, [254](#)  
COM automation, [3](#)  
COM automation, firewall settings, [16](#)  
COM version checking, [62](#)  
Comments property, [255](#)  
CompareWindow object, [118](#)  
ComputerName property, [255](#)  
Connect method, [172](#)  
Connect object, [118](#)  
connect, remote, Windows XP firewall settings, [20](#)  
ConnectSystem object, [119](#)  
CopyFile method, [173](#)  
Count example, [256](#)  
Count property, [256](#)  
CreatorName property, [259](#)

## D

Data property, [260](#)  
data ranges, [180](#)  
DataType property, [260](#)  
DataTypes and return values, [190](#)  
DCOM configuration, Workgroup simple file sharing, [37](#)  
DCOM, Windows XP firewall settings, [20](#)  
DeleteFile method, [173](#)  
Description property, [260](#)  
Differences property, [261](#)  
Distributed COM properties, logic analyzer application, [38](#)  
Distributed COM properties, logic analyzer machine-wide, [37](#)  
Distributed COM properties, remote computer application, [39](#)  
DoAction method, [174](#)  
DoCommands example, [175](#)  
DoCommands method, [174](#)  
Draw method, [178](#)

## E

EndSample property, [261](#)  
EndTime property, [262](#)  
events in Find and SimpleTrigger methods, [233](#)  
examples, LabVIEW, [75](#)  
examples, Visual Basic programs, [46](#)

Excel VB macro example, [42](#)  
Execute method, [178](#)  
Export method, [179](#)  
ExportEx method, [180](#)

## F

FaceName property, [262](#)  
File and Printer Sharing, [18](#)  
Find example, [185](#)  
Find method, [184](#)  
FindNext method, [188](#)  
FindPrev method, [189](#)  
FindResult object, [119](#)  
firewall, settings, [16](#)  
Firewall, Windows, [16, 18](#)  
folders, shared, Windows XP firewall settings, [20](#)  
Font property, [262](#)  
Found property, [263](#)  
Frame object, [120](#)  
Frame property, [263](#)  
Frames object, [120](#)  
Frames property, [263](#)

## G

Get/Put methods, Visual C++, [70](#)  
GetDataBySample method, [189](#)  
GetDataByTime method, [192](#)  
GetGroupCaption method, [193](#)  
GetInstrument method, Visual C++, [70](#)  
GetLine method, [194](#)  
GetLineLabel method, [196](#)  
GetModuleByName example, [197](#)  
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 example, [207](#)  
GoOffline method, [207](#)  
GoOnline method, [211](#)  
GoToPosition method, [212](#)

**H**

HasLegend property, 264  
HasTitle property, 264

**I**

ICMP settings, 18, 20  
ICMPv4 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 endpoint 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-ssn 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

**Q**

QueryCommand method, 220

**R**

ranges (data), specifying, 180  
RealVNC, Windows XP firewall settings, 16, 20  
RecallTriggerByFile method, 223  
RecallTriggerByName method, 224  
RecvFile 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

**S**

sample numbers, specifying a data range by, 180  
SampleBusSignalData example, 139  
SampleBusSignalData object, 138  
SampleDifference object, 149  
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



## T

TargetControlPort property, 282  
 Tcl, 84  
 TestAll method, 237  
 testing distributed COM connections, 35  
 TextColor property, 283  
 time, specifying a data range by, 180  
 TimeFound property, 283  
 TimeFoundString property, 284  
 Title property, 284  
 Tool object, 152  
 Tools example, 153  
 Tools object, 153  
 Tools property, 284  
 trademarks, 2  
 Trigger property, 285  
 troubleshooting, Distributed COM, 36  
 tutorial, LabVIEW, 72  
 type library, importing, 42, 45  
 Type property, 285

## U

using COM automation, 41

## V

Value property, 286  
 VBADisplayHelpTopic method, 237  
 VBARunMacro method, 237  
 VBARunRPCCommand method, 238  
 VBAVersion property, 287  
 VbaViewChart object, 156  
 VbaViewChartAxis object, 157  
 VbaViewChartData object, 157  
 VbaViewChartFont object, 158  
 VbaViewChartLegend object, 158  
 VbaViewChartTitle object, 159  
 VbaViewWebBrowser example, 159  
 VbaViewWebBrowser object, 159  
 VbaViewWindow example, 160  
 VbaViewWindow object, 160  
 VBE property, 287  
 version checking, COM, 62  
 Version property, 287  
 Vista, firewall settings for DCOM, 25  
 Visual Basic (in Visual Studio), 45  
 Visual Basic examples, 69  
 Visual Basic program examples, 46  
 Visual C++, 70  
 VNC, Windows XP firewall settings, 16, 20

## W

WaitComplete example, 239  
 WaitComplete method, 238  
 web server, Windows XP firewall settings, 18, 20  
 WebBrowser property, 288  
 what's changed, 293

Window object, 160  
 Windows example, 162  
 Windows Firewall, 16, 18  
 Windows Firewall with Advanced Security (Vista), 25  
 Windows object, 161  
 Windows property, 289  
 Workgroup DCOM configuration, simple file sharing, 37  
 WriteOutput method, 242

