

# COM Automation

[Online Help](#)

# Notices

© Keysight Technologies 2001-2012, 2014, 2015

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 Keysight Technologies as governed by United States and international copyright laws.

## Revision History

October 2015

Available in electronic format only

Keysight Technologies.  
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, KEYSIGHT 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. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT 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 WILL 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.

## U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at <http://www.keysight.com/find/sweula>. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data. 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 *Keysight Logic Analyzer* application includes the COM Automation Server. This software lets you write programs that control the *Keysight 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 17](#))
  - Step 2. Test your Distributed COM connection (see [page 18](#))
  - Distributed COM Troubleshooting (see [page 19](#))
- Using COM Automation (see [page 23](#))
  - Using Visual Basic for Applications (VBA) in Microsoft Excel (see [page 24](#))
  - Using Visual Basic (in Visual Studio) (see [page 26](#))
  - Example Visual Basic and Visual C++ Programs (see [page 27](#))
  - Using Visual C++ (see [page 51](#))
  - Using LabVIEW (see [page 53](#))
  - Using Perl (see [page 56](#))
  - Using Python (see [page 60](#))
  - Using Tcl (see [page 64](#))
- Reference (see [page 69](#))
  - Objects, Methods, and Properties Quick Reference (see [page 70](#))
  - Object Hierarchy Overview (see [page 80](#))
  - Objects (Quick Reference) (see [page 83](#))
  - Methods (see [page 141](#))
  - Properties (see [page 217](#))
- What's Changed (see [page 259](#))



# Contents

COM Automation—At a Glance	3
1 COM Automation Overview	
2 Setting Up for COM Automation	
Supported Networking Configurations	16
Step 1. Install the LA COM Automation client software	17
Step 2. Test your Distributed COM connection	18
Distributed COM Troubleshooting	19
To turn off simple file sharing	19
To verify logic analyzer machine-wide Distributed COM properties	19
To verify logic analyzer application Distributed COM properties	20
To verify remote computer application Distributed COM properties	21
3 Using COM Automation	
Using Visual Basic for Applications (VBA) in Microsoft Excel	24
Using Visual Basic (in Visual Studio)	26
Example Visual Basic and Visual C++ Programs	27
Loading, Running, Storing	27
Setting Up Simple Triggers	31
Setting Up Advanced Triggers	35
Changing the Sampling Mode	39
Checking the Logic Analyzer Software Version	42
Additional Visual Basic Examples	50
Using Visual C++	51
Using LabVIEW	53
Tutorial - To programmatically control the logic analyzer in LabVIEW	53
LabVIEW Examples	54
Using Perl	56
Using Python	60
Using Tcl	64

## 4 COM Automation Reference

### Objects, Methods, and Properties Quick Reference 70

#### Object Hierarchy Overview 80

#### Object Quick Reference 83

AnalyzerModule Object	84
BusSignal Object	85
BusSignalData Object	86
BusSignalDifference Object	86
BusSignalDifferences Object	86
BusSignals Object	90
CompareWindow Object	95
Connect Object	95
ConnectSystem Object	95
Exerciser Object	96
FindResult Object	97
Frame Object	97
Frames Object	98
Instrument Object	98
Marker Object	100
Markers Object	100
Module Object	105
Modules Object	105
PattgenModule Object	106
Probe Object	111
Probes Object	111
ProtocolWindow Object	114
SampleBusSignalData Object	115
SampleDifference Object	125
SampleDifferences Object	125
SelfTest Object	126
SerialModule Object	128
Tool Object	129
Tools Object	129
VbaViewChart Object	132
VbaViewChartAxis Object	133
VbaViewChartData Object	133
VbaViewChartFont Object	134
VbaViewChartLegend Object	134
VbaViewChartTitle Object	135
VbaViewWebBrowser Object	135
VbaViewWindow Object	136
Window Object	136
Windows Object	137

<b>Methods</b>	141
Add Method (BusSignals Object)	143
Add Method (Markers Object)	143
AddXML Method	143
AddPointArrays Method	144
Clear Method (for VbaViewChartData object)	145
Clear Method (for VbaViewWebBrowser object)	145
ClearOutput Method	145
Close Method	145
Connect Method	146
CopyFile Method	146
DeleteFile Method	147
DoAction Method	147
DoCommands Method	147
Draw Method	151
Execute Method	151
Export Method	152
ExportEx Method	153
Find Method	154
FindImages Method	158
FindNext Method	161
FindPrev Method	161
GetDataBySample Method	162
GetDataByTime Method	164
GetGroupCaption Method	165
GetImageList Method	165
GetLine Method	167
GetLineLabel Method	169
GetModuleByName Method	170
GetNumSamples Method	173
GetPacketCount Method	173
GetProbeByName Method	175
GetProtocolDataFields Method	175
GetRawData Method	176
GetRawTimingZoomData Method	177
GetRemoteInfo Method	178
GetSampleNumByTime Method	179
GetTime Method	179
GetToolByName Method	180
GetTriggerSampleNumber Method	180
GetValueCaption Method	181
GetWindowByName Method	181
GoOffline Method	182
GoOnline Method	186

GoToPosition Method	186
Import Method	187
ImportEx Method	187
InsertLine Method	188
IsOnline Method	188
IsTimingZoom Method	189
New Method	189
Open Method	189
PanelLock Method	190
PanelUnlock Method	193
QueryCommand Method	193
RecallTriggerByFile Method	196
RecallTriggerByName Method	196
RecvFile Method	197
Remove Method (BusSignals Object)	197
Remove Method (Markers Object)	197
RemoveAll Method	197
RemoveXML Method	198
RemoveLine Method	198
Reset Method (PattgenModule Object)	198
Resume Method (PattgenModule Object)	199
Run Method (Instrument Object)	199
Run Method (PattgenModule Object)	200
Save Method	200
SendFile Method	200
SetGroupCaption Method	201
SetLine Method	201
SetLineLabel Method	202
SetValue Method	202
SetValueArray Method	202
SetValueCaption Method	203
SimpleTrigger Method	203
Step Method	205
Stop Method (Instrument Object)	206
Stop Method (PattgenModule Object)	206
TestAll Method	206
VBADisplayHelpTopic Method	207
VBARunMacro Method	207
WaitComplete Method	207
WriteAllImagesToFiles Method	211
WriteImageToFile Method	213
WriteOutput Method	215
WriteProtocolDataFieldsToFile Method	216



<b>Properties</b>	217
Activity Property	218
Axis Property	219
AxisBase Property	219
BackgroundColor Property	220
BitSize Property	221
BitSize Property (of VbaViewChartAxis)	221
Bold Property	221
BusSignalData Property	222
BusSignalType Property	222
BusSignalDifferences Property	223
BusSignals Property	223
ByteSize Property	223
Caption Property	224
CardModels Property	224
Channels Property	224
Chart Property	225
ChartType Property	225
Color Property	226
Comments Property	226
ComputerName Property	227
Count Property	227
CreatorName Property	230
Data Property	230
DataType Property	231
Description Property	231
Differences Property	232
EndSample Property	232
EndTime Property	232
FaceName Property	233
Font Property	233
Found Property	233
Frame Property	233
Frames Property	234
HasLegend Property	234
HasTitle Property	234
Instrument Property	235
IPAddress Property	235
Item Property	235
Legend Property	236
Markers Property	237
Model Property	237
Modules Property	238
Name Property	238

NumLines Property	239
OccurrencesFound Property	239
Options Property	240
Overview Property	240
PanelLocked Property	240
Polarity Property	241
Position Property	241
Position Property (of VbaViewChartLegend)	241
Probes Property	242
Reference Property	242
RemoteComputerName Property	243
RemoteUserName Property	243
RunningStatus Property	244
SampleDifferences Property	244
SampleNum Property	244
SelfTest Property	245
Setup Property	245
Size Property	246
Slot Property	246
StartSample Property	246
StartTime Property	247
Status Property	247
StatusMsg Property	248
SubrowFound Property	248
Symbols Property	249
TargetControlPort Property	249
TextColor Property	250
TimeFound Property	250
TimeFoundString Property	250
Title Property	251
Tools Property	251
Trigger Property	251
Type Property	252
Value Property	253
VBAVersion Property	253
VBE Property	253
Version Property	254
WebBrowser Property (for VbaViewWindow object)	254
WebBrowser Property (for VbaViewWebBrowser object)	254
Windows Property	255
_NewEnum Property	255

## 5 What's Changed

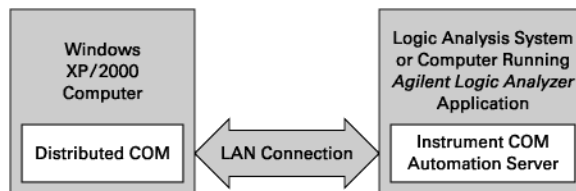
Index



# 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 *Keysight 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 80](#)).



## 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 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 16](#)) for more information.
- The logic analyzer is running version 02.00 or later of the *Keysight 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 17](#)) on your remote computer.
- 2 Test your Distributed COM connection (see [page 18](#)).

**See Also** • Using COM Automation (see [page 23](#))

## 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".

Keysight 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 *Keysight 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

-  ["Keysight Logic Analyzers Isolated Network Setup White Paper"](#)



## Step 1. Install the LA COM Automation client software

You can install either the Keysight Logic and Protocol Analyzer software package (on instruments or PCs hosting instruments) or the Keysight Logic and Protocol Analyzer COM Automation software package (on machines that do not need the full LPA software installed).

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 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\Keysight 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.

**Next** • Step 2. Test your Distributed COM connection (see [page 18](#))

## Step 2. Test your Distributed COM connection

Once you have installed Keysight Logic and Protocol Analyzer install package or the Keysight Logic and Protocol Analyzer COM Automation install package, you can run the following tools to test your COM/Distributed COM connection to the logic analyzer or diagnose COM/DCOM connection issues.

- COM testing for 64-bit, unmanaged applications/scripts - C:\Program Files\Keysight Technologies\Logic Analyzer\LA COM Automation\agTestClient.exe
- COM testing for 32-bit, unmanaged applications/scripts: C:\Program Files\Keysight Technologies\Logic Analyzer\LA COM Automation\agTestClient\_x86.exe
- COM testing for managed applications: C:\Program Files\Keysight Technologies\Logic Analyzer\COMConnectionTool.exe

If the COM Connection Tool does not resolve the problem, see also Distributed COM Troubleshooting (see [page 19](#)).

## Distributed COM Troubleshooting

If the Client Test program (see [page 18](#)) 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 *Keysight Logic Analyzer* application.
- See supported networking configurations (see [page 16](#)). 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 19](#)).

### 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 *Keysight 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 19](#))
- To verify logic analyzer application Distributed COM properties (see [page 20](#))
- To verify remote computer application Distributed COM properties (see [page 21](#))
- Make sure the logic analyzer allows DCOM access through the firewall. 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 16](#))) 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 *Keysight 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 *Keysight 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 Keysight 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, Keysight 168x/169x/169xx Logic Analyzer; then, right-click and choose Properties from the popup menu.
- 3 In the Keysight 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 Keysight 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 Keysight 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 Keysight 168x/169x/169xx Logic Analyzer and choose Properties from the popup menu.
- 3 In the Keysight 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	<p>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</b> (see page 95) object's <b>Instrument</b> (see page 235) property).</p> <p>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.</p>
Security	<p>Use default launch (and activation if on Windows XP) permissions.</p> <p>Use default access permissions.</p> <p>Use Custom configuration permissions. Verify that you have "Full Control".</p>
Endpoints	Leave at default system protocols.
Identity	<p>Normally, this tab does not appear (but it can if the <i>Keysight Logic Analyzer</i> application has been previously installed on the remote computer).</p> <p>If this tab appears, set to "The interactive user".</p>

- 4 In the Keysight 168x/169x/169xx Logic Analyzer Properties dialog, click OK.
- 5 Close the Component Services window.



# 3 Using COM Automation

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

- Using Visual Basic for Applications (VBA) in Microsoft Excel (see [page 24](#))
- Using Visual Basic (in Visual Studio) (see [page 26](#))
- Example Visual Basic Programs (see [page 27](#))
- Using Visual C++ (see [page 51](#))
- Using LabVIEW (see [page 53](#))
- Using Perl (see [page 56](#))
- Using Python (see [page 60](#))
- Using Tcl (see [page 64](#))

## 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 "Keysight 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 27](#))

## Using Visual Basic (in Visual Studio)

Before you can use the Visual Basic programming environment to control the *Keysight 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 "Keysight 168x/169x/169xx Logic Analyzer Object Library".

**See Also** • Example Visual Basic Programs (see [page 27](#))

## Example Visual Basic and Visual C++ Programs

- Loading, Running, Storing (see [page 27](#))
- Setting Up Simple Triggers (see [page 31](#))
- Setting Up Advanced Triggers (see [page 35](#))
- Changing the Sampling Mode (see [page 39](#))
- Checking the Logic Analyzer Software Version (see [page 42](#))

### See Also

- Additional Visual Basic Examples (see [page 50](#))
- Using Visual Basic for Applications (VBA) in Microsoft Excel (see [page 24](#))
- Using Visual Basic (in Visual Studio) (see [page 26](#))

### 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 *Keysight 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 Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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
// Keysight 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;

```

```
// 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 203](#)) method of the AnalyzerModule (see [page 84](#)) object.

#### Visual Basic

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 Keysight 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/Keysight 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 203](#)) method cannot set complex, multiple-step trigger sequences. To do that, you must set the AnalyzerModule (see [page 84](#)) object's Trigger (see [page 251](#)) property to an XML-format trigger specification string.

**See Also** • [Setting Up Advanced Triggers \(see page 35\)](#)

## Setting Up Advanced Triggers

This example shows how to set up advanced triggers by setting the Trigger (see [page 251](#)) property of the AnalyzerModule (see [page 84](#)) object to an XML-format trigger specification string.

```
Visual Basic ' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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 = 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);
}

```

**See Also** · [Setting Up Simple Triggers \(see page 31\)](#)

## 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 245](#)) property of the AnalyzerModule (see [page 84](#)) object.

### Visual Basic

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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
// Keysight 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/Keysight 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::IAnalyzerModulePtr pAnalyzer =
    pInst->GetModuleByName(moduleName);

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

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

// Set the state (synchronous) sampling mode.
_bstr_t myStateSamplingSetup = " \
    <Module> \
        <SamplingSetup> \
            <Sampling ChannelMode='Full' Acquisition='State' \
                AcquisitionDepth='256K' MaxSpeed='200' \
                TriggerPosition='50' /> \
            <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
End 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 Keysight 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/Keysight 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::IIstrumentPtr pInst = pConnect->GetInstrument(hostname);

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

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

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

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

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

```



```

    }
}

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

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

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

    printf("Fatal Unexpected Error:\n");
    printf("  Error Number = %08lx\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/Keysight 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/Keysight 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 235](#)) 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/Keysight 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 53](#))
- LabVIEW Examples (see [page 54](#))

### 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 53](#))
- LabVIEW 6.0 (see [page 54](#))

### 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 "Keysight 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 "Keysight 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 54](#)) 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/Keysight Technologies/Logic Analyzer/LA COM Automation/LabVIEW 6.0 Examples
- C:/Program Files/Keysight 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)
#   $LAAnalyzer      -> change to the analyzer name to transfer data
#                       from (default is the first module)
#   $LAStartRange, $LAEndRange -> change to the data range to upload
#                               (default is -10 and 10 respectively)
#
# This example was tested using ActiveState Perl version 5.6.1

# when using strict, declare *all* globals
use strict qw(vars refs subs);

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

### =====
###                               * Begin Subroutines *
### -----

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

sub PrintArrays
{
    my @arrays = @_;

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

    # Calculate the max width for each column and store in an array
    my @maxWidthArray;

    print("\n");
```



```

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

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

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

### -----
###                               End of Subroutines
### =====

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

# Logic Analyzer host name or IP address
my $LAHostNameOrIP = "localhost";

# Create the logic analyzer client server
my $LAConnect = Win32::OLE->new('AgtLA.Connect');
if (! $LAConnect)
{
    print ("Connection failed: ");
}

```

```

        print ("please install the LA COM Automation client software\n");
        exit 1;
    }

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

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

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

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

    # Get the first module's data
    my $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 $LAData = $LABusSignals->Item($index)->BusSignalData;

```

```

my $LABusSignalType = $LABusSignals->Item($index)->BusSignalType();
my $LADataType = $LAConstants->{AgtDataLong};

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

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

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

```

## Using Python

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

In both cases, you are presented with a list of objects MakePy can use to support early binding. Select Keysight 168x/169x/169xx Logic Analyzer Object Library and click OK.
- 3 Copy and paste the example code below into a file (PrintLADData.py).
- 4 Run the example from the Command Prompt by entering the command: "python PrintLADData.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.

```

COM Automation Online Help

```

LAEndRange = 10

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

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

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

        LADataArrays.append(LADataArray)

#
# Print the Arrays.
#
PrintArrays(LADataArrays)

```

## Using Tcl

- 1 Install the Tcl software.
- 2 Copy and paste the example code below into a file (PrintLADData.tcl).
- 3 Run the example from the Command Prompt by entering the command: "tclsh.exe PrintLADData.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 70](#))
- Object Hierarchy Overview (see [page 80](#))
- Objects (Quick Reference) (see [page 83](#))
- Methods (see [page 141](#))
- Properties (see [page 217](#))

## Objects, Methods, and Properties Quick Reference

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

Objects	Methods/Properties	Description
AnalyzerModule (see <a href="#">page 84</a> )		A state/timing analyzer hardware measurement module.
methods	GetRawData (see <a href="#">page 176</a> )	Given a range, returns the raw analyzer data.
	GetRawTimingZoomData (see <a href="#">page 177</a> )	Given a range, returns the raw analyzer timing zoom data.
	RecallTriggerByName (see <a href="#">page 196</a> )	Loads a named trigger from the recall buffer.
	RecallTriggerByFile (see <a href="#">page 196</a> )	Loads a previously saved trigger file on the instrument file system.
	SimpleTrigger (see <a href="#">page 203</a> )	Trigger on a simple condition with optional occurrence and storage qualification.
	WaitComplete (see <a href="#">page 207</a> )	Waits until the analyzer module, tool, and viewer measurements are complete.
properties	Setup (see <a href="#">page 245</a> )	Gets or sets the logic analyzer's XML-format setup specification.
	Trigger (see <a href="#">page 251</a> )	Gets or sets the logic analyzer's XML-format trigger specification.
BusSignal (see <a href="#">page 85</a> )		A named and grouped set of pod channels.
methods	IsTimingZoom (see <a href="#">page 189</a> )	Is this a timing zoom bus/signal?
properties	Activity (see <a href="#">page 218</a> )	Gets the activity indicators of the bus/signal.
	BitSize (see <a href="#">page 221</a> )	Gets the number of channels in the bus/signal.
	BusSignalData (see <a href="#">page 222</a> )	Gets the acquisition data associated with a bus/signal.
	BusSignalType (see <a href="#">page 222</a> )	Gets the type of bus/signal.
	ByteSize (see <a href="#">page 223</a> )	Gets the size of the bus/signal in bytes.
	Channels (see <a href="#">page 224</a> )	Gets the channels defined in the bus/signal.
	CreatorName (see <a href="#">page 230</a> )	Gets the name of the module, tool, or viewer that created this bus/signal.
	Name (see <a href="#">page 238</a> )	Gets or sets the name of the bus/signal.
	Polarity (see <a href="#">page 241</a> )	Gets the polarity of the bus/signal.
	Symbols (see <a href="#">page 249</a> )	Gets or sets the symbols associated with a bus/signal.
BusSignalData (see <a href="#">page 86</a> )		A generic bus/signal data object.
properties	Type (see <a href="#">page 252</a> )	Gets the specific bus/signal data type.
BusSignalDifference (see <a href="#">page 86</a> )		Represents the different values for a particular bus/signal within a sample that has differences.
properties	Name (see <a href="#">page 238</a> )	Gets the bus/signal name associated with the sample difference.
	Reference (see <a href="#">page 242</a> )	Gets the reference buffer value associated with the sample difference.
	Value (see <a href="#">page 253</a> )	Gets the data value associated with the sample difference.

Objects	Methods/Properties	Description
BusSignalDifferences (see <a href="#">page 86</a> )		A collection object that contains all of the SampleDifference object's buses/signals with differences.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of bus/signal differences in the collection.
	Item (see <a href="#">page 235</a> )	Given an index into the collection, gets a BusSignalDifference (see <a href="#">page 86</a> ) object from the collection.
BusSignals (see <a href="#">page 90</a> )		A collection of the hardware module's defined BusSignals.
methods	Add (see <a href="#">page 143</a> )	Adds a new bus/signal to the collection.
	Remove (see <a href="#">page 197</a> )	Removes a bus/signal from the collection.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of BusSignal (see <a href="#">page 85</a> ) objects in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the BusSignal (see <a href="#">page 85</a> ) objects in the collection given either an index or name.
CompareWindow (see <a href="#">page 95</a> )		A window that compares bus/signal data.
methods	Execute (see <a href="#">page 151</a> )	Executes the compare using the current options.
properties	Options (see <a href="#">page 240</a> )	Gets or sets the Compare window options.
	SampleDifferences (see <a href="#">page 244</a> )	Gets a collection of all the samples with differences found in the last comparison.
Connect (see <a href="#">page 95</a> )		A connection to the logic analyzer instrument.
methods	CopyFile (see <a href="#">page 146</a> )	Copies a file to the instrument file system.
	GetRemoteInfo (see <a href="#">page 178</a> )	Gets the logic analyzer's remote user login and computer name.
properties	Instrument (see <a href="#">page 235</a> )	Gets the logic analyzer instrument object.
ConnectSystem (see <a href="#">page 95</a> )		A connection to the logic analyzer system.
methods	Connect (see <a href="#">page 146</a> )	Connects to the remote logic analyzer system.
	RecvFile (see <a href="#">page 197</a> )	Copies a file from the remote logic analyzer system to your local system.
	SendFile (see <a href="#">page 200</a> )	Copies a file from your local system to the remote logic analyzer system.
Exerciser (see <a href="#">page 96</a> )		A hardware measurement module for sending defined stimulus to a DUT.
For a detailed description of its methods, refer to the <i>Using COM Commands for U4421A Module</i> section in the <i>U4421A D-PHY Analyzer and Exerciser</i> online help.		

Objects	Methods/Properties	Description
methods	ApplyToHardware(string cmd)	Applies the stimulus related settings loaded in the Setup dialog box to the stimulus hardware module.
	ExecuteCommand(string cmd, out string result)	Loads the specified stimulus settings to the Setup dialog box of the hardware module in the Logic and Protocol Analyzer GUI. In addition, the command also immediately applies these settings to the hardware module even when the module is in the Running state. The command is useful for changing stimulus settings of the module at runtime and for inserting packets dynamically.
	GetXmlFormat(string cmdKey, out string currentSettings)	Returns the XML command format for the given command name.
	LoadExerciserParameters(string cmd, out string result)	Loads the stimulus related settings for the hardware module to the Setup dialog box of the module in the Logic and Protocol Analyzer GUI.
	StartExerciser	Starts the transmission of stimulus from the hardware module to a DUT.
	StopExerciser	Stops the transmission of stimulus from the hardware module to a DUT.
FindResult (see <a href="#">page 97</a> )		Gets the results from the Find (see <a href="#">page 154</a> ), FindNext (see <a href="#">page 161</a> ), and FindPrev (see <a href="#">page 161</a> ) method calls.
properties	Found (see <a href="#">page 233</a> )	Gets the found status.
	OccurrencesFound (see <a href="#">page 239</a> )	Gets the number of occurrences found.
	SubrowFound (see <a href="#">page 248</a> )	Gets the subrow number if found on a subrow.
	TimeFound (see <a href="#">page 250</a> )	Gets the time found as a double.
	TimeFoundString (see <a href="#">page 250</a> )	Gets the time found as a string.
Frame (see <a href="#">page 97</a> )		A logic analyzer frame.
properties	ComputerName (see <a href="#">page 227</a> )	Gets the computer name of the logic analyzer frame.
	Description (see <a href="#">page 231</a> )	Gets a description of the logic analyzer frame.
	IPAddress (see <a href="#">page 235</a> )	Gets the frame's IP address(es).
	TargetControlPort (see <a href="#">page 249</a> )	Gets or sets the target control port value.
Frames (see <a href="#">page 98</a> )		A collection of logic analyzer frames connected via the multiframe connector.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of frames in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the frames in the collection given either an index, computer name, or IP address.
Instrument (see <a href="#">page 98</a> )		The logic analyzer instrument.



Objects	Methods/Properties	Description
methods	Close (see <a href="#">page 145</a> )	Closes the current configuration.
	DeleteFile (see <a href="#">page 147</a> )	Deletes a file on the instrument file system.
	DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
	Export (see <a href="#">page 152</a> )	Exports data to a file on the instrument file system.
	ExportEx (see <a href="#">page 153</a> )	Exports data to a file on the instrument file system.
	GetProbeByName (see <a href="#">page 175</a> )	Given a probe name, returns its corresponding probe object.
	GetModuleByName (see <a href="#">page 170</a> )	Given a module name, returns its corresponding hardware module object.
	GetToolByName (see <a href="#">page 180</a> )	Given a tool name, returns its corresponding tool object.
	GetWindowByName (see <a href="#">page 181</a> )	Given a window name, returns its corresponding window object.
	Import (see <a href="#">page 187</a> )	Imports data from a file located on the instrument file system.
	ImportEx (see <a href="#">page 187</a> )	Imports data from a file located on the instrument file system into a particular module.
	GoOffline (see <a href="#">page 182</a> )	Disconnects the user interface from the logic analyzer frame.
	GoOnline (see <a href="#">page 186</a> )	Connects the user interface to a specific logic analyzer frame.
	IsOnline (see <a href="#">page 188</a> )	Tells whether the user interface is connected to a logic analyzer frame.
	New (see <a href="#">page 189</a> )	Creates a new instrument Overview.
	Open (see <a href="#">page 189</a> )	Loads a previously saved configuration file on the instrument file system.
	PanelLock (see <a href="#">page 190</a> )	Disables user access to the instrument front panel or remote display.
	PanelUnlock (see <a href="#">page 193</a> )	Re-enables user access to the instrument front panel or remote display.
	QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
	Run (see <a href="#">page 199</a> )	Starts running all modules.
	Save (see <a href="#">page 200</a> )	Saves the current configuration to a file on the instrument file system.
	Stop (see <a href="#">page 206</a> )	Stops all currently running modules.
	VBADisplayHelpTopic (see <a href="#">page 207</a> )	Displays the help page and topic for an installed VBA project.
	VBARunMacro (see <a href="#">page 207</a> )	Runs the specified VBA macro as if that macro was selected in the Macros dialog box.
	WaitComplete (see <a href="#">page 207</a> )	Waits until all module, tool, and viewer measurements are complete.

Objects	Methods/Properties	Description
properties	Frames (see <a href="#">page 234</a> )	Gets a collection of logic analyzer frames connected via the multiframe connector.
	Markers (see <a href="#">page 237</a> )	Gets a collection of all the markers in the instrument.
	Model (see <a href="#">page 237</a> )	Gets the model number.
	Modules (see <a href="#">page 238</a> )	Gets a collection of all the hardware modules in the instrument.
	Overview (see <a href="#">page 240</a> )	Gets the XML-format Overview window specification.
	PanelLocked (see <a href="#">page 240</a> )	Indicates the front panel is locked. If locked, the message is returned.
	Probes (see <a href="#">page 242</a> )	Gets a collection of all currently defined probes.
	RemoteComputerName (see <a href="#">page 243</a> )	Gets or sets the remote computer name.
	RemoteUserName (see <a href="#">page 243</a> )	Gets or sets the remote user login name.
	SelfTest (see <a href="#">page 245</a> )	Gets the SelfTest object.
	Status (see <a href="#">page 247</a> )	Gets the status of all hardware modules.
	Tools (see <a href="#">page 251</a> )	Gets a collection of all active software tools.
	VBAVersion (see <a href="#">page 253</a> )	Gets the version number of VBA.
	VBE (see <a href="#">page 253</a> )	Gets the VBE extensibility object.
	Version (see <a href="#">page 254</a> )	Gets the version number of the system software.
	Windows (see <a href="#">page 255</a> )	Gets a collection of all active windows.
Marker (see <a href="#">page 100</a> )		A reference point in the captured data.
properties	Comments (see <a href="#">page 226</a> )	Gets or sets the marker comments.
	Name (see <a href="#">page 238</a> )	Gets or sets the name of the marker.
	TextColor (see <a href="#">page 250</a> )	Gets or sets the marker text color.
	BackgroundColor (see <a href="#">page 220</a> )	Gets or sets the marker background color.
	Position (see <a href="#">page 241</a> )	Gets or sets the marker position.
Markers (see <a href="#">page 100</a> )		A collection of all the defined markers.
methods	Add (see <a href="#">page 143</a> )	Adds a new marker to the collection using specific values.
	AddXML (see <a href="#">page 143</a> )	Adds multiple markers to the collection using an XML string.
	Remove (see <a href="#">page 197</a> )	Removes a marker from the collection.
	RemoveAll (see <a href="#">page 197</a> )	Removes all markers from the collection.
	RemoveXML (see <a href="#">page 198</a> )	Removes multiple markers from the collection using an XML string.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of markers in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the markers in the collection given either an index or name.
Module (see <a href="#">page 105</a> )		A generic hardware module.

Objects	Methods/Properties	Description
methods	DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
	WaitComplete (see <a href="#">page 207</a> )	Waits until the module, tool, and viewer measurements are complete.
properties	BusSignals (see <a href="#">page 223</a> )	Gets a collection of the module's defined bus/signals.
	CardModels (see <a href="#">page 224</a> )	Gets the card model numbers.
	Description (see <a href="#">page 231</a> )	Gets a description of the module.
	Frame (see <a href="#">page 233</a> )	Gets the frame in which the module resides.
	Model (see <a href="#">page 237</a> )	Gets the model number.
	Name (see <a href="#">page 238</a> )	Gets or sets the name of the module.
	RunningStatus (see <a href="#">page 244</a> )	Gets the detailed running status of the module.
	Slot (see <a href="#">page 246</a> )	Gets the module's slot location in the frame.
	Status (see <a href="#">page 247</a> )	Gets the status of the module.
	StatusMsg (see <a href="#">page 248</a> )	Gets the formatted status message.
	Type (see <a href="#">page 252</a> )	Gets the specific module type.
Modules (see <a href="#">page 105</a> )		A collection of all the hardware modules installed in the instrument.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of modules in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the modules in a collection given either a slot, index, or name.
PattgenModule (see <a href="#">page 106</a> )		A pattern generator hardware module.
methods	GetLine (see <a href="#">page 167</a> )	Gets an instruction or vector at line number.
	GetLineLabel (see <a href="#">page 169</a> )	Gets a vector's label value at line number.
	InsertLine (see <a href="#">page 188</a> )	Inserts a new instruction or vector after line number.
	RemoveLine (see <a href="#">page 198</a> )	Removes the instruction or vector at line number.
	Reset (see <a href="#">page 198</a> )	Resets the current line number to the first line.
	Resume (see <a href="#">page 199</a> )	Resumes running the pattern generator from the current line number.
	Run (see <a href="#">page 200</a> )	Starts running the pattern generator.
	SetLine (see <a href="#">page 201</a> )	Sets an instruction or vector at line number.
	SetLineLabel (see <a href="#">page 202</a> )	Sets a vector's label value at line number.
	Step (see <a href="#">page 205</a> )	Steps the pattern generator from the current line number.
	Stop (see <a href="#">page 206</a> )	Stops the pattern generator if it is currently running.
properties	NumLines (see <a href="#">page 239</a> )	Gets the number of lines in the main sequence.

Objects	Methods/Properties	Description
Probe (see <a href="#">page 111</a> )		A generic probe.
methods	DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
properties	Name (see <a href="#">page 238</a> )	Gets or sets the name of the probe.
	Type (see <a href="#">page 252</a> )	Gets the probe's type.
Probes (see <a href="#">page 111</a> )		A collection of all active probes.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of probes in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the probes in the collection given either an index or name.
ProtocolWindow (see <a href="#">page 114</a> )		A window which displays protocol data from a serial analyzer.
methods	GetProtocolDataFields (see <a href="#">page 175</a> )	Gets the acquisition data for the fields displayed in the Protocol Viewer.
	WriteProtocolDataFieldsToFile (see <a href="#">page 216</a> )	Writes the acquisition data displayed in various fields in the Protocol Viewer to a specified CSV file.
	GetTriggerSampleNumber (see <a href="#">page 180</a> )	Gets the packet number and channel of the trigger packet.
	FindImages (see <a href="#">page 158</a> )	Searches acquisition data for frames containing images and puts them into the Protocol Viewer's list of images.
	GetImageList (see <a href="#">page 165</a> )	Returns a string containing the Protocol Viewer's list of images. Optionally, it writes this list to a text file.
	GetPacketCount (see <a href="#">page 173</a> )	Given a channel name, returns the number of packets for that particular channel.
	WriteAllImagesToFiles (see <a href="#">page 211</a> )	Writes all the images in the Protocol Viewer's list to separate bitmap files in a specified folder.
	WriteImageToFile (see <a href="#">page 213</a> )	Writes a single image to a bitmap file. The image is selected using the 'Frame Number' string from the image list.
SampleBusSignalData (see <a href="#">page 115</a> )		The data for a specific bus/signal captured in the state or timing sampling modes.
methods	GetDataBySample (see <a href="#">page 162</a> )	Given a sample range, returns an array of data.
	GetDataByTime (see <a href="#">page 164</a> )	Given a time range, returns an array of data.
	GetNumSamples (see <a href="#">page 173</a> )	Given a range, returns the number of samples stored.
	GetSampleNumByTime (see <a href="#">page 179</a> )	Gets the closest sample number corresponding to the time given.
	GetTime (see <a href="#">page 179</a> )	Given a range, returns the time for this bus/signal in the format specified by the data type given.

Objects	Methods/Properties	Description
properties	DataType (see <a href="#">page 231</a> )	Gets the recommended bus/signal data type.
	EndSample (see <a href="#">page 232</a> )	Gets the data's ending sample number relative to trigger.
	EndTime (see <a href="#">page 232</a> )	Gets the data's ending time relative to trigger.
	StartSample (see <a href="#">page 246</a> )	Gets the data's starting sample number relative to trigger.
	StartTime (see <a href="#">page 247</a> )	Gets the data's starting time relative to trigger.
SampleDifference (see <a href="#">page 125</a> )		Represents a sample containing a CompareWindow difference.
properties	BusSignalDifferences (see <a href="#">page 223</a> )	Gets a collection of all the buses/signals with differences for this sample.
	SampleNum (see <a href="#">page 244</a> )	Gets the sample number at which differences occurred.
SampleDifferences (see <a href="#">page 125</a> )		A collection object of all the sample differences in the CompareWindow object.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of bus/signal differences in the collection.
	Item (see <a href="#">page 235</a> )	Given an index into the collection, gets a SampleDifference (see <a href="#">page 125</a> ) object from the collection.
SelfTest (see <a href="#">page 126</a> )		Object for running instrument self-tests.
methods	TestAll (see <a href="#">page 206</a> )	Runs an instrument's self-tests.
SerialModule (see <a href="#">page 128</a> )		A hardware measurement module for viewing and analyzing serial data.
methods	RecallTriggerByFile (see <a href="#">page 196</a> )	Loads a previously saved trigger file located on the instrument file system.
Tool (see <a href="#">page 129</a> )		A generic instrument software tool.
methods	DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
	QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
properties	BusSignals (see <a href="#">page 223</a> )	Gets a collection of the tool's defined bus/signals.
	Name (see <a href="#">page 238</a> )	Gets or sets the name of the tool.
	Type (see <a href="#">page 252</a> )	Gets the specific tool type.
Tools (see <a href="#">page 129</a> )		A collection of all of the instrument's software tools.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of tools in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the tools in the collection given either an index or name.
VbaViewChart (see <a href="#">page 132</a> )		A chart for the VbaViewWindow.
methods	Draw (see <a href="#">page 151</a> )	Draws the chart.

Objects	Methods/Properties	Description
properties	Axis (see <a href="#">page 219</a> )	Gets the chart axis given an axis type.
	ChartType (see <a href="#">page 225</a> )	Gets or sets the chart type.
	Data (see <a href="#">page 230</a> )	Gets the chart data.
	HasLegend (see <a href="#">page 234</a> )	Gets or sets if the legend is visible.
	HasTitle (see <a href="#">page 234</a> )	Gets or sets if the title is visible.
	Legend (see <a href="#">page 236</a> )	Gets the chart legend.
	Title (see <a href="#">page 251</a> )	Gets the title of the chart.
VbaViewChartAxis (see <a href="#">page 133</a> )		The axis of a VbaViewChart.
properties	AxisBase (see <a href="#">page 219</a> )	Gets or sets the chart axis base.
	BitSize (see <a href="#">page 221</a> )	Gets or sets the width of the data in bits. This is used to format the Axis values.
	HasTitle (see <a href="#">page 234</a> )	Gets or sets if the title is visible.
	Title (see <a href="#">page 251</a> )	Gets the title of the axis.
VbaViewChartData (see <a href="#">page 133</a> )		The data of a VbaViewChart.
methods	AddPointArrays (see <a href="#">page 144</a> )	Adds an array of points to the chart. This is only valid for chart types AgtChartTypeLine and AgtChartTypeXYScatter.
	Clear (see <a href="#">page 145</a> )	Clears all of the chart data.
	GetGroupCaption (see <a href="#">page 165</a> )	Gets the caption associated with a group (row).
	GetValueCaption (see <a href="#">page 181</a> )	Gets the caption associated with all values at index (column).
	SetGroupCaption (see <a href="#">page 201</a> )	Sets the caption associated with a group (row).
	SetValue (see <a href="#">page 202</a> )	Sets an individual value in the chart array.
	SetValueArray (see <a href="#">page 202</a> )	Sets an array of values in the chart array starting at index 0.
	SetValueCaption (see <a href="#">page 203</a> )	Sets the caption associated with all values at index (column).
VbaViewChartFont (see <a href="#">page 134</a> )		The font of a VbaViewChartTitle.
properties	Bold (see <a href="#">page 221</a> )	Gets or sets the text thickness.
	Color (see <a href="#">page 226</a> )	Gets or sets the text color.
	FaceName (see <a href="#">page 233</a> )	Gets or sets the text face name string.
	Size (see <a href="#">page 246</a> )	Gets or sets the text size.
VbaViewChartLegend (see <a href="#">page 134</a> )		The legend of a VbaViewChart.
properties	Position (see <a href="#">page 241</a> )	Gets or sets the chart legend position.
VbaViewChartTitle (see <a href="#">page 135</a> )		The title of a VbaViewChart.
properties	Caption (see <a href="#">page 224</a> )	Gets or sets the chart title caption.
	Font (see <a href="#">page 233</a> )	Gets the chart title font.
VbaViewWebBrowser (see <a href="#">page 135</a> )		A web browser for the VbaViewWindow.
methods	Clear (see <a href="#">page 145</a> )	Displays an empty web page.

Objects	Methods/Properties	Description
properties	WebBrowser (see <a href="#">page 254</a> )	Gets the contained IWebBrowser2 interface.
VbaViewWindow (see <a href="#">page 136</a> )		A window for Visual Basic for Applications (VBA) program output.
methods	ClearOutput (see <a href="#">page 145</a> )	Clears the strings from the output window.
	WriteOutput (see <a href="#">page 215</a> )	Writes a string to the output window.
properties	Chart (see <a href="#">page 225</a> )	Gets the Chart view.
	WebBrowser (see <a href="#">page 254</a> )	Gets the Web Browser view.
Window (see <a href="#">page 136</a> )		A generic instrument display window.
methods	DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
	DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
	Find (see <a href="#">page 154</a> )	Finds a specified data event with optional occurrence and time duration.
	FindNext (see <a href="#">page 161</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 161</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 186</a> )	Moves the center of the window to a new position.
	QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
properties	BusSignals (see <a href="#">page 223</a> )	Gets a collection of the window's defined bus/signals.
	Name (see <a href="#">page 238</a> )	Gets or sets the name of the window.
	Type (see <a href="#">page 252</a> )	Gets the window's type.
Windows (see <a href="#">page 137</a> )		A collection of all of the instrument's display windows.
properties	_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
	Count (see <a href="#">page 227</a> )	Gets the number of windows in the collection.
	Item (see <a href="#">page 235</a> )	Gets one of the windows in the collection given either an index or name.

**See Also** · [Logic Analyzer Object Hierarchy Overview \(see page 80\)](#)

## 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 105](#)) (that represent the hardware installed in the instrument), Probes (see [page 111](#)) (that organize probes connected to a DUT), Tools (see [page 129](#)) (that filter or decode captured data), and Windows (see [page 137](#)) (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 105](#)), Tool (see [page 129](#)), and Window (see [page 136](#)) objects return data through a BusSignalData (see [page 86](#)) object, which returns information about:

- Directly acquired data when obtained from a Module (see [page 105](#)) object.
- Created and/or manipulated data when obtained from a Tool (see [page 129](#)) object.
- Displayed data when obtained from a Window (see [page 136](#)) object.

The following tree illustrates the hierarchy of the Instrument object:

- Connect (see [page 95](#)) – Not used in integrated VBA environment.
  - Instrument (see [page 98](#)) – Not used in integrated VBA environment.
    - Frames (see [page 98](#))
      - Frame (see [page 97](#))
    - Markers (see [page 100](#))
      - Marker (see [page 100](#))
    - SelfTest (see [page 126](#))
    - Modules (see [page 105](#)) – Collection of all modules in the system.
      - *Module* (see [page 105](#)) – Generic module object.
        - BusSignals (see [page 90](#))
          - BusSignal (see [page 85](#))
            - *BusSignalData* (see [page 86](#))
            - *SampleBusSignalData* (see [page 115](#))
        - AnalyzerModule (see [page 84](#)) – Logic analyzer specific object.
        - PattgenModule (see [page 106](#)) – Pattern generator specific object.
        - SerialModule (see [page 128](#)) – Serial analyzer specific object.
        - Exerciser (see [page 96](#)) – A hardware measurement module for sending defined stimulus to a DUT.
      - Probes (see [page 111](#))
        - Probe (see [page 111](#))
      - Tools (see [page 129](#))
        - Tool (see [page 129](#))
          - BusSignals (see [page 90](#))
            - BusSignal (see [page 85](#))
              - *BusSignalData* (see [page 86](#)) – Generic bus/signal data object.
              - *SampleBusSignalData* (see [page 115](#)) – Sample bus/signal data specific object.



- Windows (see [page 137](#))
  - *Window* (see [page 136](#)) – Generic window object.
    - BusSignals (see [page 90](#))
      - BusSignal (see [page 85](#))
        - *BusSignalData* (see [page 86](#))
        - *SampleBusSignalData* (see [page 115](#))
    - FindResult (see [page 97](#))
  - CompareWindow (see [page 95](#)) – Compare window specific object.
    - SampleDifferences (see [page 125](#))
      - SampleDifference (see [page 125](#))
        - BusSignalDifferences (see [page 86](#))
          - BusSignalDifference (see [page 86](#))
  - ProtocolWindow (see [page 114](#)) – Protocol Viewer window specific object.
  - VbaViewWindow (see [page 136](#)) – VbaView window specific object.
    - VbaViewChart (see [page 132](#))
      - VbaViewChartAxis (see [page 133](#))
      - VbaViewChartData (see [page 133](#))
      - VbaViewChartLegend (see [page 134](#))
      - VbaViewChartTitle (see [page 135](#))
        - VbaViewChartFont (see [page 134](#))
    - VbaViewWebBrowser (see [page 135](#))
- ConnectSystem (see [page 95](#))

*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 105](#)) object is generic; it contains the methods and properties that are common to both the AnalyzerModule (see [page 84](#)) object and the PattgenModule (see [page 106](#)) object.
- The AnalyzerModule (see [page 84](#)) object contains logic analyzer specific properties and methods (such as the GetDataBySample (see [page 162](#)) method), but it also has access to all of the generic properties and methods in the Module (see [page 105](#)) object.
- The PattgenModule (see [page 106](#)) object contains pattern generator specific properties and methods (such as the InsertLine (see [page 188](#)) method), but it also has access to all of the generic properties and methods in the Module (see [page 105](#)) object.

If you know what type of object you have, use the specific objects (like AnalyzerModule (see [page 84](#)) and PattgenModule (see [page 106](#))).

If you don't know what type of object you have, use the generic object (like Module (see [page 105](#))); 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 83](#))

## Object Quick Reference

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

Object	Description
<a href="#">AnalyzerModule</a> (see <a href="#">page 84</a> )	A state/timing analyzer hardware measurement module.
<a href="#">BusSignal</a> (see <a href="#">page 85</a> )	A named and grouped set of pod channels.
<a href="#">BusSignalData</a> (see <a href="#">page 86</a> )	A generic bus/signal data object.
<a href="#">BusSignalDifference</a> (see <a href="#">page 86</a> )	Represents the different values for a particular bus/signal within a sample that has differences.
<a href="#">BusSignalDifferences</a> (see <a href="#">page 86</a> )	A collection object that contains all of the <a href="#">SampleDifference</a> object's buses/signals with differences.
<a href="#">BusSignals</a> (see <a href="#">page 90</a> )	A collection of the hardware module's defined <a href="#">BusSignals</a> .
<a href="#">CompareWindow</a> (see <a href="#">page 95</a> )	A window that compares bus/signal data.
<a href="#">Connect</a> (see <a href="#">page 95</a> )	A connection to the logic analyzer instrument.
<a href="#">ConnectSystem</a> (see <a href="#">page 95</a> )	A connection to the logic analyzer system.
<a href="#">Exerciser</a> (see <a href="#">page 96</a> )	A hardware measurement module for sending defined stimulus to a DUT.
<a href="#">FindResult</a> (see <a href="#">page 97</a> )	Gets the results from the <a href="#">Find</a> (see <a href="#">page 154</a> ), <a href="#">FindNext</a> (see <a href="#">page 161</a> ), and <a href="#">FindPrev</a> (see <a href="#">page 161</a> ) method calls.
<a href="#">Frame</a> (see <a href="#">page 97</a> )	A logic analyzer frame.
<a href="#">Frames</a> (see <a href="#">page 98</a> )	A collection of logic analyzer frames connected via the multiframe connector.
<a href="#">Instrument</a> (see <a href="#">page 98</a> )	The logic analyzer instrument.
<a href="#">Marker</a> (see <a href="#">page 100</a> )	A reference point in the captured data.
<a href="#">Markers</a> (see <a href="#">page 100</a> )	A collection of all the defined markers.
<a href="#">Module</a> (see <a href="#">page 105</a> )	A generic hardware module.
<a href="#">Modules</a> (see <a href="#">page 105</a> )	A collection of all the hardware modules installed in the instrument.
<a href="#">PattgenModule</a> (see <a href="#">page 106</a> )	A pattern generator hardware module.
<a href="#">Probe</a> (see <a href="#">page 111</a> )	A generic probe.
<a href="#">Probes</a> (see <a href="#">page 111</a> )	A collection of all active probes.
<a href="#">ProtocolWindow</a> (see <a href="#">page 114</a> )	A window which displays protocol data from a serial analyzer.
<a href="#">SampleBusSignalData</a> (see <a href="#">page 115</a> )	The data for a specific bus/signal captured in the state or timing sampling modes.
<a href="#">SampleDifference</a> (see <a href="#">page 125</a> )	Represents a sample containing a <a href="#">CompareWindow</a> difference.
<a href="#">SampleDifferences</a> (see <a href="#">page 125</a> )	A collection object of all the sample differences in the <a href="#">CompareWindow</a> object.
<a href="#">SelfTest</a> (see <a href="#">page 126</a> )	Provides methods for running instrument self-tests.
<a href="#">SerialModule</a> (see <a href="#">page 128</a> )	A hardware measurement module for viewing and analyzing serial data.
<a href="#">Tool</a> (see <a href="#">page 129</a> )	A generic instrument software tool.
<a href="#">Tools</a> (see <a href="#">page 129</a> )	A collection of all of the instrument's software tools.

Object	Description
VbaViewChart (see <a href="#">page 132</a> )	A chart for the VbaViewWindow.
VbaViewChartAxis (see <a href="#">page 133</a> )	The axis of a VbaViewChart.
VbaViewChartData (see <a href="#">page 133</a> )	The data of a VbaViewChart.
VbaViewChartFont (see <a href="#">page 134</a> )	The font properties of a VbaViewChartTitle.
VbaViewChartLegend (see <a href="#">page 134</a> )	The legend of a VbaViewChart.
VbaViewChartTitle (see <a href="#">page 135</a> )	The title of a VbaViewChart.
VbaViewWebBrowser (see <a href="#">page 135</a> )	A web browser for the VbaViewWindow.
VbaViewWindow (see <a href="#">page 136</a> )	A window for Visual Basic for Applications (VBA) program output.
Window (see <a href="#">page 136</a> )	A generic instrument display window.
Windows (see <a href="#">page 137</a> )	A collection of all of the instrument's display windows.

**See Also** • [Logic Analyzer Object Hierarchy Overview \(see page 80\)](#)

AnalyzerModule Object

[ [Automation Home \(see page 3\)](#) ] [ [Objects \(see page 83\)](#) ]

**To Access** •

```
Dim variable As AgtLA.AnalyzerModule
Set variable = Module (see page 105)
```

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

Since this object is derived from the Module (see [page 105](#)) object, a Set must be used to get to these specific methods and properties. For example:

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 176</a> )	Given a range, returns the raw analyzer data.
GetRawTimingZoomData (see <a href="#">page 177</a> )	Given a range, returns the raw analyzer timing zoom data.
RecallTriggerByName (see <a href="#">page 196</a> )	Loads a named trigger from the recall buffer.
RecallTriggerByFile (see <a href="#">page 196</a> )	Loads a previously saved trigger file on the instrument file system.
SimpleTrigger (see <a href="#">page 203</a> )	Trigger on a simple condition with optional occurrence and storage qualification.
WaitComplete (see <a href="#">page 207</a> )	Waits until the analyzer module, tool, and viewer measurements are complete.

(Also Includes Module (see [page 105](#)) object methods)

## Properties

Property	Description
Setup (see <a href="#">page 245</a> )	Gets or sets the logic analyzer's XML-format setup specification.
Trigger (see <a href="#">page 251</a> )	Gets or sets the logic analyzer's XML-format trigger specification.

(Also Includes Module (see [page 105](#)) object properties)

## BusSignal Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 91](#)) ]

## To Access

- BusSignals.Item (see [page 235](#)) IndexOrName
- BusSignals IndexOrName

## Methods

Method	Description
IsTimingZoom (see <a href="#">page 189</a> )	Is this a timing zoom bus/signal?

## Properties

Property	Description
Activity (see <a href="#">page 218</a> )	Gets the activity indicators of the bus/signal.
BitSize (see <a href="#">page 221</a> )	Gets the number of channels in the bus/signal.
BusSignalData (see <a href="#">page 222</a> )	Gets the acquisition data associated with a bus/signal.
BusSignalType (see <a href="#">page 222</a> )	Gets the type of bus/signal.
ByteSize (see <a href="#">page 223</a> )	Gets the size of the bus/signal in bytes.
Channels (see <a href="#">page 224</a> )	Gets the channels defined in the bus/signal.
CreatorName (see <a href="#">page 230</a> )	Gets the name of the module, tool, or viewer that created this bus/signal.
Name (see <a href="#">page 238</a> )	Gets or sets the name of the bus/signal.
Polarity (see <a href="#">page 241</a> )	Gets the polarity of the bus/signal.
Symbols (see <a href="#">page 249</a> )	Gets or sets the symbols associated with a bus/signal.

**See Also** • BusSignals (see [page 90](#)) object

BusSignalData Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**To Access** • BusSignal.BusSignalData (see [page 222](#))

**Methods** There are no methods.

### Properties

Property	Description
Type (see <a href="#">page 252</a> )	Gets the specific bus/signal data type.

**See Also** • BusSignalData (see [page 222](#)) property

BusSignalDifference Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**To Access** • BusSignalDifferences.Item (see [page 235](#)) IndexOrName  
• BusSignalDifferences IndexOrName

**Methods** There are no methods.

### Properties

Property	Description
Name (see <a href="#">page 238</a> )	Gets the bus/signal name associated with the sample difference.
Reference (see <a href="#">page 242</a> )	Gets the reference buffer value associated with the sample difference.
Value (see <a href="#">page 253</a> )	Gets the data value associated with the sample difference.

**See Also** • BusSignalDifferences (see [page 86](#)) object

BusSignalDifferences Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**To Access** • SampleDifference.BusSignalDifferences (see [page 223](#))

**Methods** There are no methods.

### Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of bus/signal differences in the collection.
Item (see <a href="#">page 235</a> )	Given an index into the collection, gets a BusSignalDifference (see <a href="#">page 86</a> ) object from the collection.

**See Also** • BusSignalDifferences (see [page 223](#)) property

### BusSignalDifferences Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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);

// 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 83](#)) ] [ Example (see [page 91](#)) ]

- To Access**
- Module.BusSignals (see [page 223](#))
  - Tool.BusSignals (see [page 223](#))
  - Window.BusSignals (see [page 223](#))

## Methods

Method	Description
Add (see <a href="#">page 143</a> )	Adds a new bus/signal to the collection.
Remove (see <a href="#">page 197</a> )	Removes a bus/signal from the collection.

## Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of BusSignal (see <a href="#">page 85</a> ) objects in the collection.
Item (see <a href="#">page 235</a> )	Gets one of the BusSignal (see <a href="#">page 85</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 259](#))).

**See Also** · BusSignals (see [page 223](#)) property

### BusSignals Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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
    End Select
Next myBusSignal
```

```

        Case AgtBusSignalProbed
            myString = myString + "AgtBusSignalProbed" + vbNewLine
            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 Keysight 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/Keysight 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);

        // 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 83](#)) ] [ Example (see [page 87](#)) ]

### To Access

```

Dim variable As AgtLA.CompareWindow
Set variable = Window (see page 136)

```

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

### Methods

Method	Description
Execute (see <a href="#">page 151</a> )	Executes the compare using the current options.

(Also Includes Window (see [page 136](#)) methods)

### Properties

Property	Description
Options (see <a href="#">page 240</a> )	Gets or sets the Compare window options.
SampleDifferences (see <a href="#">page 244</a> )	Gets a collection of all the samples with differences found in the last comparison.

(Also Includes Window (see [page 136](#)) properties)

### Connect Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

### To Access

```

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

```

### Methods

Method	Description
CopyFile (see <a href="#">page 146</a> )	Copies a file to the instrument file system.
GetRemoteInfo (see <a href="#">page 178</a> )	Gets the logic analyzer's remote user login and computer name.

### Properties

Property	Description
Instrument (see <a href="#">page 235</a> )	Gets the logic analyzer instrument object.

### ConnectSystem Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

### To Access

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

## Methods

Method	Description
Connect (see <a href="#">page 146</a> )	Connects to the remote logic analyzer system.
RecvFile (see <a href="#">page 197</a> )	Copies a file from the remote logic analyzer system to your local system.
SendFile (see <a href="#">page 200</a> )	Copies a file from your local system to the remote logic analyzer system.

**Properties** There are no properties.

Exerciser Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

## To Access

```
Dim variable As AgtLA.Exerciser
Set variable = Module (see page 105)
```

**Description** The Exerciser object represents a hardware measurement module for sending defined stimulus to a DUT.

Since this object is derived from the Module (see [page 105](#)) object, a Set must be used to get to these specific methods and properties. For example:

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Exerciser specific object.
Dim myExerciser As AgtLA.Exerciser
Set myExerciser = myInst.Modules(0)
```

## Methods

The Exerciser object provides generic methods that accept command string parameters.

The object is currently used for sending D-PHY stimulus from the U4421A D-PHY Analyzer and Exerciser module to a D-PHY DUT.

For a detailed description of each of these methods and the XML format command string parameters that these methods accept, refer to the *Using COM Commands for U4421A Module* section in the *U4421A D-PHY Analyzer and Exerciser online help*.



Method	Description
ApplyToHardware(string cmd)	Applies the stimulus related settings loaded in the Setup dialog box to the stimulus hardware module.
ExecuteCommand(string cmd, out string result)	Loads the specified stimulus settings to the Setup dialog box of the hardware module in the Logic and Protocol Analyzer GUI. In addition, the command also immediately applies these settings to the hardware module even when the module is in the Running state. The command is useful for changing stimulus settings of the module at runtime and for inserting packets dynamically.
GetXmlFormat(string cmdKey, out string currentSettings)	Returns the XML command format for the given command name.
LoadExerciserParameters(string cmd, out string result)	Loads the stimulus related settings for the hardware module to the Setup dialog box of the module in the Logic and Protocol Analyzer GUI.
StartExerciser	Starts the transmission of stimulus from the hardware module to a DUT.
StopExerciser	Stops the transmission of stimulus from the hardware module to a DUT.

(Also Includes Module (see [page 105](#)) object methods)

## FindResult Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 155](#)) ]

- To Access**
- Window.Find(Event) (see [page 154](#))
  - Window.FindNext() (see [page 161](#))
  - Window.FindPrev() (see [page 161](#))

**Methods** There are no methods.

## Properties

Property	Description
Found (see <a href="#">page 233</a> )	Gets the found status.
OccurrencesFound (see <a href="#">page 239</a> )	Gets the number of occurrences found.
SubrowFound (see <a href="#">page 248</a> )	Gets the subrow number if found on a subrow.
TimeFound (see <a href="#">page 250</a> )	Gets the time found as a double.
TimeFoundString (see <a href="#">page 250</a> )	Gets the time found as a string.

- See Also**
- Find (see [page 154](#)) method
  - FindNext (see [page 161](#)) method
  - FindPrev (see [page 161](#)) method
  - GetDataByTime (see [page 164](#)) method

## Frame Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

- To Access**
- Frames.Item (see [page 235](#)) IndexOrName
  - Frames.IndexOrName

**Methods** There are no methods.

### Properties

Property	Description
ComputerName (see <a href="#">page 227</a> )	Gets the computer name of the logic analyzer frame.
Description (see <a href="#">page 231</a> )	Gets a description of the logic analyzer frame.
IPAddress (see <a href="#">page 235</a> )	Gets the frame's IP address(es).
TargetControlPort (see <a href="#">page 249</a> )	Gets or sets the target control port value.

**See Also** • Frames (see [page 98](#)) object

Frames Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**To Access** • Instrument.Frames (see [page 234](#))

**Methods** There are no methods.

### Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of For Each ... Next.
Count (see <a href="#">page 227</a> )	Gets the number of frames in the collection.
Item (see <a href="#">page 235</a> )	Gets one of the frames in the collection given either an index, computer name, or IP address.

**See Also** • Frames (see [page 234](#)) property

Instrument Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**To Access** • When using Visual Basic outside of the *Keysight Logic Analyzer* application:

```
Dim variable As AgtLA.Instrument
Set variable = Connect.Instrument (see page 235) HostNameOrIpAddress
```

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

### Methods

Method	Description
Close (see <a href="#">page 145</a> )	Closes the current configuration.
DeleteFile (see <a href="#">page 147</a> )	Deletes a file on the instrument file system.
DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
Export (see <a href="#">page 152</a> )	Exports data to a file on the instrument file system.

Method	Description
ExportEx (see <a href="#">page 153</a> )	Exports data to a file on the instrument file system.
GetProbeByName (see <a href="#">page 175</a> )	Given a probe name, returns its corresponding probe object.
GetModuleByName (see <a href="#">page 170</a> )	Given a module name, returns its corresponding hardware module object.
GetToolByName (see <a href="#">page 180</a> )	Given a tool name, returns its corresponding tool object.
GetWindowByName (see <a href="#">page 181</a> )	Given a window name, returns its corresponding window object.
Import (see <a href="#">page 187</a> )	Imports data from a file located on the instrument file system.
ImportEx (see <a href="#">page 187</a> )	Imports data from a file located on the instrument file system into a particular module.
GoOffline (see <a href="#">page 182</a> )	Disconnects the user interface from the logic analyzer frame.
GoOnline (see <a href="#">page 186</a> )	Connects the user interface to a specific logic analyzer frame.
IsOnline (see <a href="#">page 188</a> )	Tells whether the user interface is connected to a logic analyzer frame.
New (see <a href="#">page 189</a> )	Creates a new instrument Overview.
Open (see <a href="#">page 189</a> )	Loads a previously saved configuration file on the instrument file system.
PanelLock (see <a href="#">page 190</a> )	Disables user access to the instrument front panel or remote display.
PanelUnlock (see <a href="#">page 193</a> )	Re-enables user access to the instrument front panel or remote display.
QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
Run (see <a href="#">page 199</a> )	Starts running all modules.
Save (see <a href="#">page 200</a> )	Saves the current configuration to a file on the instrument file system.
Stop (see <a href="#">page 206</a> )	Stops all currently running modules.
VBADisplayHelpTopic (see <a href="#">page 207</a> )	Displays the help page and topic for an installed VBA project.
VBARunMacro (see <a href="#">page 207</a> )	Runs the specified VBA macro as if that macro was selected in the Macros dialog box.
WaitComplete (see <a href="#">page 207</a> )	Waits until all module, tool, and viewer measurements are complete.

## Properties

Property	Description
Frames (see <a href="#">page 234</a> )	Gets a collection of logic analyzer frames connected via the multiframe connector.
Markers (see <a href="#">page 237</a> )	Gets a collection of all the markers in the instrument.
Model (see <a href="#">page 237</a> )	Gets the model number.
Modules (see <a href="#">page 238</a> )	Gets a collection of all the hardware modules in the instrument.
Overview (see <a href="#">page 240</a> )	Gets the XML-format Overview window specification.
PanelLocked (see <a href="#">page 240</a> )	Indicates the front panel is locked. If locked, the message is returned.
Probes (see <a href="#">page 242</a> )	Gets a collection of all currently defined probes.
RemoteComputerName (see <a href="#">page 243</a> )	Gets or sets the remote computer name.
RemoteUserName (see <a href="#">page 243</a> )	Gets or sets the remote user login name.
SelfTest (see <a href="#">page 245</a> )	Gets the SelfTest object.
Status (see <a href="#">page 247</a> )	Gets the status of all hardware modules.

Property	Description
Tools (see <a href="#">page 251</a> )	Gets a collection of all active software tools.
VBAVersion (see <a href="#">page 253</a> )	Gets the version number of VBA.
VBE (see <a href="#">page 253</a> )	Gets the VBE extensibility object.
Version (see <a href="#">page 254</a> )	Gets the version number of the system software.
Windows (see <a href="#">page 255</a> )	Gets a collection of all active windows.

**See Also** • Instrument (see [page 235](#)) property

Marker Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 101](#)) ]

**To Access** • Markers.Item (see [page 235](#)) IndexOrName  
• Markers IndexOrName

**Methods** There are no methods.

**Properties**

Property	Description
Comments (see <a href="#">page 226</a> )	Gets or sets the marker comments.
Name (see <a href="#">page 238</a> )	Gets or sets the name of the marker.
TextColor (see <a href="#">page 250</a> )	Gets or sets the marker text color.
BackgroundColor (see <a href="#">page 220</a> )	Gets or sets the marker background color.
Position (see <a href="#">page 241</a> )	Gets or sets the marker position.

**See Also** • Markers (see [page 100](#)) object

Markers Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 101](#)) ]

**To Access** • Instrument.Markers (see [page 237](#))

**Methods**

Method	Description
Add (see <a href="#">page 143</a> )	Adds a new marker to the collection using specific values.
AddXML (see <a href="#">page 143</a> )	Adds multiple markers to the collection using an XML string.
Remove (see <a href="#">page 197</a> )	Removes a marker from the collection.
RemoveAll (see <a href="#">page 197</a> )	Removes all markers from the collection.
RemoveXML (see <a href="#">page 198</a> )	Removes multiple markers from the collection using an XML string.

## Properties

Property	Description
<code>_NewEnum</code> (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
<code>Count</code> (see <a href="#">page 227</a> )	Gets the number of markers in the collection.
<code>Item</code> (see <a href="#">page 235</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 237](#)) property

### Markers Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 + vbNewLine + 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 Keysight 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/Keysight 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);

        // 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 83](#)) ]

- To Access**
- Instrument.GetModuleByName(Name) (see [page 170](#))
  - Modules.Item (see [page 235](#)) IndexOrName
  - Modules.IndexOrName

## Methods

Method	Description
DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.
WaitComplete (see <a href="#">page 207</a> )	Waits until the module's measurement completes.

## Properties

Property	Description
BusSignals (see <a href="#">page 223</a> )	Gets a collection of the module's defined bus/signals.
Description (see <a href="#">page 231</a> )	Gets a description of the module.
Frame (see <a href="#">page 233</a> )	Gets the frame in which the module resides.
Model (see <a href="#">page 237</a> )	Gets the model number.
Name (see <a href="#">page 238</a> )	Gets or sets the name of the module.
RunningStatus (see <a href="#">page 244</a> )	Gets the detailed running status of the module.
Slot (see <a href="#">page 246</a> )	Gets the module's slot location in the frame.
Status (see <a href="#">page 247</a> )	Gets the status of the module.
StatusMsg (see <a href="#">page 248</a> )	Gets the formatted status message.
Type (see <a href="#">page 252</a> )	Gets the specific module type.

- See Also**
- Modules (see [page 105](#)) object

## Modules Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

- To Access**
- Instrument.Modules (see [page 238](#))

**Methods** There are no methods.

## Properties

Property	Description
<code>_NewEnum</code> (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
<code>Count</code> (see <a href="#">page 227</a> )	Gets the number of modules in the collection.
<code>Item</code> (see <a href="#">page 235</a> )	Gets one of the modules in a collection given either a slot, index, or name.

## See Also

- `Modules` (see [page 238](#)) property
- `Module` (see [page 105](#)) object
- `GetModuleByName` (see [page 170](#)) method

## PattgenModule Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

## To Access

•

```
Dim variable As AgtLA.PattgenModule
Set variable = Module (see page 105)
```

## Description

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

Since this object is derived from the `Module` (see [page 105](#)) object, a `Set` must be used to get to these specific methods and properties. For example:

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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
<code>GetLine</code> (see <a href="#">page 167</a> )	Gets an instruction or vector at line number.
<code>GetLineLabel</code> (see <a href="#">page 169</a> )	Gets a vector's label value at line number.
<code>InsertLine</code> (see <a href="#">page 188</a> )	Inserts a new instruction or vector after line number.
<code>RemoveLine</code> (see <a href="#">page 198</a> )	Removes the instruction or vector at line number.
<code>Reset</code> (see <a href="#">page 198</a> )	Resets the current line number to the first line.
<code>Resume</code> (see <a href="#">page 199</a> )	Resumes running the pattern generator from the current line number.
<code>Run</code> (see <a href="#">page 200</a> )	Starts running the pattern generator.

Method	Description
SetLine (see <a href="#">page 201</a> )	Sets an instruction or vector at line number.
SetLineLabel (see <a href="#">page 202</a> )	Sets a vector's label value at line number.
Step (see <a href="#">page 205</a> )	Steps the pattern generator from the current line number.
Stop (see <a href="#">page 206</a> )	Stops the pattern generator if it is currently running.

(Also Includes Module (see [page 105](#)) object methods)

## Properties

Property	Description
NumLines (see <a href="#">page 239</a> )	Gets the number of lines in the main sequence.

(Also Includes Module (see [page 105](#)) object properties)

## PattgenModule Example

**Visual Basic**

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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\\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 83](#)) ] [ Example (see [page 111](#)) ]

- To Access**
- Instrument.GetProbeByName(Name) (see [page 175](#))
  - Probes.Item (see [page 235](#)) IndexOrName
  - Probes.IndexOrName

#### Methods

Method	Description
DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.

#### Properties

Property	Description
Name (see <a href="#">page 238</a> )	Gets or sets the name of the probe.
Type (see <a href="#">page 252</a> )	Gets the probe's type.

- See Also**
- Probes (see [page 111](#)) object

#### Probes Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 111](#)) ]

- To Access**
- Instrument.Probes (see [page 242](#))

**Methods** There are no methods.

#### Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of probes in the collection.
Item (see <a href="#">page 235</a> )	Gets one of the probes in the collection given either an index or name.

- See Also**
- Probes (see [page 242](#)) property
  - Probe (see [page 111](#)) object
  - GetProbeByName (see [page 175](#)) method

#### Probes Example

**Visual Basic**

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 Keysight 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/Keysight 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 = 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);
}

```

#### ProtocolWindow Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

#### To Access

```

Dim variable As AgtLA.ProtocolWindow
Set variable = Window (see page 136)

```

**Description** The ProtocolWindow object represents an output window for viewing protocol data from a serial analyzer.

#### Methods

Method	Description
GetProtocolDataFields (see <a href="#">page 175</a> )	Gets the acquisition data for the fields displayed in the Protocol Viewer.
WriteProtocolDataFieldsToFile (see <a href="#">page 216</a> )	Writes the acquisition data displayed in various fields in the Protocol Viewer to a specified CSV file.
GetTriggerSampleNumber (see <a href="#">page 180</a> )	Gets the packet number and channel of the trigger packet.
FindImages (see <a href="#">page 158</a> )	Searches acquisition data for frames containing images and puts them into the Protocol Viewer's list of images.
GetImageList (see <a href="#">page 165</a> )	Returns a string containing the Protocol Viewer's list of images. Optionally, it writes this list to a text file.

Method	Description
GetPacketCount (see <a href="#">page 173</a> )	Given a channel name, returns the number of packets for that particular channel.
WriteAllImagesToFiles (see <a href="#">page 211</a> )	Writes all the images in the Protocol Viewer's list to separate bitmap files in a specified folder.
WriteImageToFile (see <a href="#">page 213</a> )	Writes a single image to a bitmap file. The image is selected using the 'Frame Number' string from the image list.

(Also includes Window object's methods)

### SampleBusSignalData Object

[ [Automation Home](#) (see [page 3](#)) ] [ [Objects](#) (see [page 83](#)) ] [ [Example](#) (see [page 116](#)) ]

#### To Access

```
Dim variable As AgtLA.SampleBusSignalData
Set variable = BusSignalData (see page 86)
```

#### Description

The SampleBusSignalData object represents the data associated with a bus/signal. The data can be uploaded using the methods GetDataBySample (see [page 162](#)) and GetDataByTime (see [page 164](#)).

Since this object is derived from the BusSignalData (see [page 86](#)) object, a Set must be used to get to these specific methods and properties. For example:

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 162</a> )	Given a sample range, returns an array of data.
GetDataByTime (see <a href="#">page 164</a> )	Given a time range, returns an array of data.
GetNumSamples (see <a href="#">page 173</a> )	Given a range, returns the number of samples stored.
GetSampleNumByTime (see <a href="#">page 179</a> )	Gets the closest sample number corresponding to the time given.
GetTime (see <a href="#">page 179</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 231</a> )	Gets the recommended bus/signal data type.
EndSample (see <a href="#">page 232</a> )	Gets the data's ending sample number relative to trigger.
EndTime (see <a href="#">page 232</a> )	Gets the data's ending time relative to trigger.
StartSample (see <a href="#">page 246</a> )	Gets the data's starting sample number relative to trigger.
StartTime (see <a href="#">page 247</a> )	Gets the data's starting time relative to trigger.

### SampleBusSignalData Example

**Visual Basic**

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 Keysight 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/Keysight 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 83](#)) ] [ Example (see [page 87](#)) ]

**To Access**

- SampleDifferences.Item (see [page 235](#)) IndexOrName
- SampleDifferences IndexOrName

**Methods** There are no methods.

#### Properties

Property	Description
BusSignalDifferences (see <a href="#">page 223</a> )	Gets a collection of all the buses/signals with differences for this sample.
SampleNum (see <a href="#">page 244</a> )	Gets the sample number at which differences occurred.

**See Also**

- SampleDifferences (see [page 125](#)) object

#### SampleDifferences Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**To Access**

- CompareWindow.SampleDifferences (see [page 244](#))

**Methods** There are no methods.

#### Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of bus/signal differences in the collection.
Item (see <a href="#">page 235</a> )	Given an index into the collection, gets a SampleDifference (see <a href="#">page 125</a> ) object from the collection.

**See Also** • SampleDifferences (see [page 244](#)) property

SelfTest Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 126](#)) ]

**To Access** • Instrument.SelfTest (see [page 245](#))

## Methods

Method	Description
TestAll (see <a href="#">page 206</a> )	Runs an instrument's self-tests.

**Properties** There are no properties.

**See Also** • SelfTest (see [page 245](#)) 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 Keysight 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/Keysight 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::IIstrumentPtr 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);
}

```

#### SerialModule Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

#### To Access

```

Dim variable As AgtLA.SerialModule
Set variable = Module (see page 105)

```

#### Description

The SerialModule object represents a hardware measurement module for viewing and analyzing serial data.

Since this object is derived from the Module (see [page 105](#)) object, a Set must be used to get to these specific methods and properties. For example:

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 SerialModule specific object.
Dim myAnalyzer As AgtLA.SerialModule
Set myAnalyzer = myInst.Modules(0)
MsgBox "Trigger: " + myAnalyzer.Trigger

```



## Methods

Method	Description
RecallTriggerByFile (see <a href="#">page 196</a> )	Loads a previously saved trigger file located on the instrument file system.

(Also Includes Module (see [page 105](#)) object methods)

### Tool Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 129](#)) ]

## To Access

- Instrument.GetToolByName(Name) (see [page 180](#))
- Tools.Item (see [page 235](#)) IndexOrName
- Tools IndexOrName

## Methods

Method	Description
DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.

## Properties

Property	Description
BusSignals (see <a href="#">page 223</a> )	Gets a collection of the tool's defined bus/signals.
Name (see <a href="#">page 238</a> )	Gets or sets the name of the tool.
Type (see <a href="#">page 252</a> )	Gets the specific tool type.

## See Also

- Tools (see [page 129](#)) object

### Tools Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 129](#)) ]

## To Access

- Instrument.Tools (see [page 251](#))

## Methods

There are no methods.

## Properties

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of tools in the collection.
Item (see <a href="#">page 235</a> )	Gets one of the tools in the collection given either an index or name.

## See Also

- Tools (see [page 251](#)) property
- Tool (see [page 129](#)) object
- GetToolByName (see [page 180](#)) method

### Tools Example

**Visual Basic**

```

' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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
    MsgBox "MPC8XX IA Properties: " + XMLCmdResponse
End If

```

**Visual C++**

```

//
// This simple Visual C++ Console application demonstrates how to use
// the Keysight 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/Keysight 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);

            // 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 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewWindow.Chart (see [page 225](#))

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

### Methods

Method	Description
Draw (see <a href="#">page 151</a> )	Draws the chart.

### Properties

Property	Description
Axis (see <a href="#">page 219</a> )	Gets the chart axis given an axis type.
ChartType (see <a href="#">page 225</a> )	Gets or sets the chart type.
Data (see <a href="#">page 230</a> )	Gets the chart data.
HasLegend (see <a href="#">page 234</a> )	Gets or sets if the legend is visible.
HasTitle (see <a href="#">page 234</a> )	Gets or sets if the title is visible.
Legend (see <a href="#">page 236</a> )	Gets the chart legend.
Title (see <a href="#">page 251</a> )	Gets the title of the chart.

**See Also** · Chart (see [page 225](#)) property

### VbaViewChartAxis Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewChart.Axis (see [page 219](#))

**Description** The VbaViewChartAxis object represents a chart in the VbaViewWindow.

### Methods

There are no methods.

### Properties

Property	Description
AxisBase (see <a href="#">page 219</a> )	Gets or sets the chart axis base.
BitSize (see <a href="#">page 221</a> )	Gets or sets the width of the data in bits. This is used to format the Axis values.
HasTitle (see <a href="#">page 234</a> )	Gets or sets if the title is visible.
Title (see <a href="#">page 251</a> )	Gets the title of the axis.

**See Also** · Axis (see [page 219](#)) property

### VbaViewChartData Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewChart.Data (see [page 230](#))

**Description** The VbaViewChartData object represents the data in a VbaViewChart.

**Methods**

Method	Description
AddPointArrays (see <a href="#">page 144</a> )	Adds an array of points to the chart. This is only valid for chart types AgtChartTypeLine and AgtChartTypeXYScatter.
Clear (see <a href="#">page 145</a> )	Clears all of the chart data.
GetGroupCaption (see <a href="#">page 165</a> )	Gets the caption associated with a group (row).
GetValueCaption (see <a href="#">page 181</a> )	Gets the caption associated with all values at index (column).
SetGroupCaption (see <a href="#">page 201</a> )	Sets the caption associated with a group (row).
SetValue (see <a href="#">page 202</a> )	Sets an individual value in the chart array.
SetValueArray (see <a href="#">page 202</a> )	Sets an array of values in the chart array starting at index 0.
SetValueCaption (see <a href="#">page 203</a> )	Sets the caption associated with all values at index (column).

**Properties** There are no properties.

**See Also** · Data (see [page 230](#)) property

VbaViewChartFont Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewChartTitle.Font (see [page 233](#))

**Description** The VbaViewChartFont object represents the font of a VbaViewChartTitle.

**Methods** There are no methods.

**Properties**

Property	Description
Bold (see <a href="#">page 221</a> )	Gets or sets the text thickness.
Color (see <a href="#">page 226</a> )	Gets or sets the text color.
FaceName (see <a href="#">page 233</a> )	Gets or sets the text face name string.
Size (see <a href="#">page 246</a> )	Gets or sets the text size.

**See Also** · Font (see [page 233](#)) property

VbaViewChartLegend Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewChart.Legend (see [page 236](#))

**Description** The VbaViewChartLegend object represents the legend in a VbaViewChart.

**Methods** There are no methods.

**Properties**

Property	Description
Position (see <a href="#">page 241</a> )	Gets or sets the chart legend position.

**See Also** · Legend (see [page 236](#)) property

VbaViewChartTitle Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**To Access** · VbaViewChart.Title (see [page 251](#))

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

**Methods** There are no methods.

#### Properties

Property	Description
Caption (see <a href="#">page 224</a> )	Gets or sets the chart title caption.
Font (see <a href="#">page 233</a> )	Gets the chart title font.

**See Also** · Title (see [page 251](#)) property

VbaViewWebBrowser Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 135](#)) ]

**To Access** · VbaViewWindow.WebBrowser (see [page 254](#))

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

#### Methods

Method	Description
Clear (see <a href="#">page 145</a> )	Displays an empty web page.

#### Properties

Property	Description
WebBrowser (see <a href="#">page 254</a> )	Gets the contained IWebBrowser2 interface.

**Requirements** · Version (see [page 42](#)): 3.20 or later.

**See Also** · WebBrowser (see [page 254](#)) property

#### VbaViewWebBrowser Example

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

' Display the Keysight home page.
browser.Navigate ("http://www.keysight.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 83](#)) ] [ Example (see [page 136](#)) ]

**To Access**

·

```
Dim variable As AgtLA.VbaViewWindow
Set variable = Window (see page 136)
```

**Description**

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

**Methods**

Method	Description
ClearOutput (see <a href="#">page 145</a> )	Clears the strings from the output window.
WriteOutput (see <a href="#">page 215</a> )	Writes a string to the output window.

(Also Includes Window (see [page 136](#)) methods)

**Properties**

Property	Description
Chart (see <a href="#">page 225</a> )	Gets the Chart view.
WebBrowser (see <a href="#">page 254</a> )	Gets the Web Browser view.

(Also Includes Window (see [page 136](#)) 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 83](#)) ] [ Example (see [page 137](#)) ]

**To Access**

- Instrument.GetWindowByName(Name) (see [page 181](#))
- Windows.Item (see [page 235](#)) IndexOrName
- Windows.IndexOrName

**Methods**

Method	Description
DoAction (see <a href="#">page 147</a> )	Execute a specific XML-based command action.
DoCommands (see <a href="#">page 147</a> )	Execute a particular XML-based command.
Find (see <a href="#">page 154</a> )	Finds a specified data event with optional occurrence and time duration.
FindNext (see <a href="#">page 161</a> )	Finds the next event by searching forward from the last event found using the event specified by the last call to Find (see <a href="#">page 154</a> ).
FindPrev (see <a href="#">page 161</a> )	Finds the previous event by searching backward from the last event found using the event specified by the last call to Find (see <a href="#">page 154</a> ).
GoToPosition (see <a href="#">page 186</a> )	Moves the center of the window to a new position.
QueryCommand (see <a href="#">page 193</a> )	Query for XML-based commands.



**Properties**

Property	Description
BusSignals (see <a href="#">page 223</a> )	Gets a collection of the window's defined bus/signals.
Name (see <a href="#">page 238</a> )	Gets or sets the name of the window.
Type (see <a href="#">page 252</a> )	Gets the window's type.

**See Also** · Windows (see [page 137](#)) object

## Windows Object

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 137](#)) ]

**To Access** · Instrument.Windows (see [page 255](#))

**Methods** There are no methods.

**Properties**

Property	Description
_NewEnum (see <a href="#">page 255</a> )	Used by Visual Basic to support the implementation of <b>For Each ... Next</b> .
Count (see <a href="#">page 227</a> )	Gets the number of windows in the collection.
Item (see <a href="#">page 235</a> )	Gets one of the windows in the collection given either an index or name.

**See Also** · Windows (see [page 255](#)) property  
 · Window (see [page 136](#)) object  
 · GetWindowByName (see [page 181](#)) method

## Windows Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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\\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 143](#))
- Add Method (Markers Object) (see [page 143](#))
- AddXML Method (see [page 143](#))
- AddPointArrays Method (see [page 144](#))
- Clear Method (see [page 145](#)) (for VbaViewChartData object)
- Clear Method (see [page 145](#)) (for VbaViewWebBrowser object)
- ClearOutput Method (see [page 145](#))
- Close Method (see [page 145](#))
- Connect Method (see [page 146](#))
- CopyFile Method (see [page 146](#))
- DeleteFile Method (see [page 147](#))
- DoAction Method (see [page 147](#))
- DoCommands Method (see [page 147](#))
- Draw Method (see [page 151](#))
- Execute Method (see [page 151](#))
- Export Method (see [page 152](#))
- ExportEx Method (see [page 153](#))
- Find Method (see [page 154](#))
- FindImages Method (see [page 158](#))
- FindNext Method (see [page 161](#))
- FindPrev Method (see [page 161](#))
- GetDataBySample Method (see [page 162](#))
- GetDataByTime Method (see [page 164](#))
- GetGroupCaption Method (see [page 165](#))
- GetImageList Method (see [page 165](#))
- GetLine Method (see [page 167](#))
- GetLineLabel Method (see [page 169](#))
- GetModuleByName Method (see [page 170](#))
- GetNumSamples Method (see [page 173](#))
- GetPacketCount Method (see [page 173](#))
- GetProbeByName Method (see [page 175](#))
- GetProtocolDataFields Method (see [page 175](#))
- GetRawData Method (see [page 176](#))
- GetRawTimingZoomData Method (see [page 177](#))
- GetRemoteInfo Method (see [page 178](#))
- GetSampleNumByTime Method (see [page 179](#))
- GetTime Method (see [page 179](#))
- GetToolByName Method (see [page 180](#))
- GetTriggerSampleNumber Method (see [page 180](#))
- GetValueCaption Method (see [page 181](#))
- GetWindowByName Method (see [page 181](#))
- GoOffline Method (see [page 182](#))

- GoOnline Method (see [page 186](#))
- GoToPosition Method (see [page 186](#))
- Import Method (see [page 187](#))
- ImportEx Method (see [page 187](#))
- InsertLine Method (see [page 188](#))
- IsOnline Method (see [page 188](#))
- IsTimingZoom Method (see [page 189](#))
- New Method (see [page 189](#))
- Open Method (see [page 189](#))
- PanelLock Method (see [page 190](#))
- PanelUnlock Method (see [page 193](#))
- QueryCommand Method (see [page 193](#))
- RecallTriggerByFile Method (see [page 196](#))
- RecallTriggerByName Method (see [page 196](#))
- RecvFile Method (see [page 197](#))
- Remove Method (BusSignals Object) (see [page 197](#))
- Remove Method (Markers Object) (see [page 197](#))
- RemoveAll Method (see [page 197](#))
- RemoveXML Method (see [page 198](#))
- RemoveLine Method (see [page 198](#))
- Reset Method (see [page 198](#))
- Resume Method (see [page 199](#))
- Run Method (Instrument Object) (see [page 199](#))
- Run Method (PattgenModule Object) (see [page 200](#))
- Save Method (see [page 200](#))
- SendFile Method (see [page 200](#))
- SetGroupCaption Method (see [page 201](#))
- SetLine Method (see [page 201](#))
- SetLineLabel Method (see [page 202](#))
- SetValue Method (see [page 202](#))
- SetValueArray Method (see [page 202](#))
- SetValueCaption Method (see [page 203](#))
- SimpleTrigger Method (see [page 203](#))
- Step Method (see [page 205](#))
- Stop Method (Instrument Object) (see [page 206](#))
- Stop Method (PattgenModule Object) (see [page 206](#))
- TestAll Method (see [page 206](#))
- VBADisplayHelpTopic Method (see [page 207](#))
- VBARunMacro Method (see [page 207](#))
- WaitComplete Method (see [page 207](#))
- WriteAllImagesToFiles Method (see [page 211](#))
- WriteImageToFile Method (see [page 213](#))
- WriteOutput Method (see [page 215](#))
- WriteProtocolDataFieldsToFile Method (see [page 216](#))

## Add Method (BusSignals Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • BusSignals (see [page 90](#)) object**Description** Adds a BusSignal (see [page 85](#)) object to the BusSignals (see [page 90](#)) 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 90</a> ) object.
Name	A <b>String</b> containing the name of the bus/signal to be added.
Channels	<p>A <b>String</b> containing <i>MultiplePodChannels</i> or the <b>String</b> "None" if no channels are assigned. <b>MultiplePodChannels</b> contains a comma separated list of <i>PodChannels</i>. <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><b>PodChannels Example:</b>  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><b>MultiplePodChannels Example:</b>  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 83](#)) ] [ Example (see [page 101](#)) ]**Applies To** • Markers (see [page 100](#)) object**Description** Adds a Marker (see [page 100](#)) object to the Markers (see [page 100](#)) 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 100</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 83](#)) ] [ Example (see [page 101](#)) ]

**Applies To** · Markers (see [page 100](#)) object

**Description** Adds multiple Marker (see [page 100](#)) objects to the Markers (see [page 100](#)) 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 100</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 83](#)) ]

**Applies To** · VbaViewChartData (see [page 133](#)) 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 133</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:

AgtDataPointSize	Enum Value	Description
AgtDataPointSizeSmall	1	
AgtDataPointSizeMedium	2	
AgtDataPointSizeLarge	3	

Clear Method (for VbaViewChartData object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartData (see [page 133](#)) object

**Description** Clears all of the chart data.

**VB Syntax** object.Clear

Parameters	Definition
object	An expression that evaluates to a VbaViewChartData (see <a href="#">page 133</a> ) object.

Clear Method (for VbaViewWebBrowser object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewWebBrowser (see [page 135](#)) 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 135</a> ) object.

**Requirements** • Version (see [page 42](#)): 3.20 or later.

ClearOutput Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • VbaViewWindow (see [page 136](#)) 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 136</a> ) object.

Close Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 83](#)) ] [ Example ]

**Applies To** • ConnectSystem (see [page 95](#)) object

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

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

Parameters	Definition
object	An expression that evaluates to a ConnectSystem (see <a href="#">page 95</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>Keysight Logic Analyzer</i> application will run. This parameter is optional.

**See Also** • RecvFile (see [page 197](#)) method  
• SendFile (see [page 200](#)) method

#### CopyFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Connect (see [page 95](#)) object

**Description** Copies a file to the instrument file system.

The Instrument (see [page 235](#)) 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 Connect (see <a href="#">page 95</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 235](#)) property

#### DeleteFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) object  
 • Module (see [page 105](#)) object  
 • Probe (see [page 111](#)) object  
 • Tool (see [page 129](#)) object  
 • Window (see [page 136](#)) 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 83](#)) ] [ Example (see [page 148](#)) ]

**Applies To** • Instrument (see [page 98](#)) object

- Module (see [page 105](#)) object
- Probe (see [page 111](#)) object
- Tool (see [page 129](#)) object
- Window (see [page 136](#)) 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 98</a> )	A string containing the XML format "<Configuration> element" (in the online help).
Module (see <a href="#">page 105</a> )	A string containing the XML format "<Module> element" (in the online help) (under Configuration Setup).
Probe (see <a href="#">page 111</a> )	A string containing the XML format "<Probe> element" (in the online help) (under Configuration Setup).
Tool (see <a href="#">page 129</a> )	A string containing the XML format "<Tool> element" (in the online help) (under Configuration Setup).
Window (see <a href="#">page 136</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 Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 Keysight 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/Keysight 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);

// 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 = 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);
}

```

#### Draw Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChart (see [page 132](#)) object

**Description** Draws the chart.

**VB Syntax** object.Draw

Parameters	Definition
object	An expression that evaluates to a VbaViewChart (see <a href="#">page 132</a> ) object.

#### Execute Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** • CompareWindow (see [page 95](#)) 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 95</a> ) object.

- See Also**
- Differences (see [page 232](#)) property
  - SampleDifferences (see [page 244](#)) property

#### Export Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

- Applies To**
- Instrument (see [page 98](#)) 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 98</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 152</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 153](#)) method.

- See Also**
- ExportEx (see [page 153](#)) method
  - Import (see [page 187](#)) method
  - ImportEx (see [page 187](#)) method

#### Data Ranges

Data ranges are specified by Variant start and end parameters.

For:	Use the Variant Type:
sample numbers	<b>Integer or 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 152](#)) method
- GetNumSamples (see [page 173](#)) method
- GetTime (see [page 179](#)) method

## ExportEx Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</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. 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 "<module_name>:TimingZoom", for example, "My 16950A-1:TimingZoom". When the "Pattern Generator CSV text file" FileType is specified, the SourceName must be a pattern generator module.
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.

Parameters	Definition
StartRange EndRange	<b>Variants</b> specifying the data range (see <a href="#">page 152</a> ).
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>Keysight 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> </ul>
FileOptions	<p>An XML format <b>String</b> that lets you specify export options (as you can with the <i>Keysight 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 <code>&lt;Options&gt;</code> element attribute values can be:</p> <ul style="list-style-type: none"> <li>▪ <code>SeparationCharacters</code> - 'string'</li> <li>▪ <code>WriteLineNumberColumn</code> - 'F' (false) or 'T' (true)</li> <li>▪ <code>LineNumberColumnName</code> - 'string'</li> <li>▪ <code>IncludeHeader</code> - 'F' (false) or 'T' (true)</li> <li>▪ <code>WriteFixedWidthColumns</code> - 'F' (false) or 'T' (true)</li> </ul> <p>Certain file types support certain file options (as in the <i>Keysight Logic Analyzer</i> application):</p> <ul style="list-style-type: none"> <li>▪ "Standard CSV text file" - <code>SeparationCharacters</code>, <code>WriteLineNumberColumns</code>, and <code>LineNumberColumnName</code>.</li> <li>▪ "Module CSV text file" - <code>SeparationCharacters</code>, <code>WriteLineNumberColumns</code>, <code>LineNumberColumnName</code>, and <code>IncludeHeader</code>.</li> <li>▪ "Module binary file" - <code>IncludeHeader</code>.</li> <li>▪ "Pattern Generator CSV text file" - <code>SeparationCharacters</code>.</li> </ul>

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

**See Also**

- Export (see [page 152](#)) method
- Import (see [page 187](#)) method
- ImportEx (see [page 187](#)) method

#### Find Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 155](#)) ]

**Applies To**

- Window (see [page 136](#)) 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 136</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 204</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.
When	A <b>String</b> specifying a time duration or other operator. <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> Note that these strings are case-sensitive.
Duration	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: <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> Examples: <ul style="list-style-type: none"> <li>▪ "1 ps"</li> <li>▪ "1ns..30ns"</li> </ul>

**Return Value** A FindResult (see [page 97](#)) object containing the results of the find.

**See Also**

- FindNext (see [page 161](#)) method
- FindPrev (see [page 161](#)) method
- FindResult (see [page 97](#)) 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 Keysight 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/Keysight 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);

        // 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);
}

```

#### FindImages Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**NOTE:** This method is applicable only for those Logic and Protocol Analyzer modules that support the Image View.

**Applies To** ProtocolWindow (see [page 114](#)) object

**Description** Searches acquisition data for frames containing images and puts them into the Protocol Viewer's list of images.

**VB Syntax** object.**FindImages** start end errorMessage success

Parameters	Definition
object	An expression that evaluates a ProtocolWindow object.
start	A <b>VARIANT</b> which specifies the starting point in the search of the acquisition data. If start is a floating point number, then start is interpreted as a packet time. If start is a string, then start must be one of these case-sensitive strings. <ul style="list-style-type: none"> <li>▪ "Beginning Of Data"</li> <li>▪ "End Of Data"</li> <li>▪ "Trigger"</li> <li>▪ "M1"</li> <li>▪ "M2" or any other marker name</li> </ul>
end	A <b>VARIANT</b> which specifies the ending point in the search of the acquisition data. If end is a floating point number, then end is interpreted as a packet time. If end is a string, then end must be one of these case-sensitive strings. <ul style="list-style-type: none"> <li>▪ "Beginning Of Data"</li> <li>▪ "End Of Data"</li> <li>▪ "Trigger"</li> <li>▪ "M1"</li> <li>▪ "M2" or any other marker name</li> </ul>
errorMessage	A string output parameter. If success returns VARIANT_FALSE, then errorMessage contains information about the error that occurred. If success returns VARIANT_TRUE, then errorMessage is the empty string.
success	A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE. VARIANT_FALSE indicates an error occurred and errorMessage contains an explanation of the error.

**Remarks** FindImages is one of the several methods that enables the COM user to identify and extract image frame data. It performs the same function as the 'Find' button on the 'Image View' tab of the Protocol viewer.

The Protocol Viewer maintains an internal list of image frames that have been found in the acquisition data. Both FindImages and the 'Find' button update the same internal list. All the other COM commands which access image data use this internal list. Thus, a call to FindImages is the first step to access image data.

#### FindImages Example

**Visual C++** **NOTE:** The example below is incomplete. The example does not contain code which acquires data. In order to work, the code in the example must have packet data in acquisition memory. It is also required that the packet data must contain some image frames.

```
// Create the Connect object.

AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));...

// Using the Connect object, obtain the IInstrument interface.

AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostName);...

// Using the IInstrument interface, obtain the IWindows interface

AgtLA::IWindowsPtr windows = pInst->GetWindows();...
```

```

// Loop through all the windows in the instrument looking for the first
window

// that presents an IProtocolWindow interface.
AgtLA::IprotoWindowPtr protocolWindow = NULL;
for (long i=0; i < windows->GetCount(); i++)
{
    // Using the IWindows interface, obtain a IWindow interface.
    AgtLA::IWindowPtr window = windows->GetItem(_variant_t(i));

    // Check if this window is a ProtocolWindow by type-casting the
    // module pointer.
    protocolWindow= windows->GetItem(_variant_t(i));
    if(protocolWindow != NULL)
    {
        // Found a ProtocolWindow
        break;
    }
}
if (protocolWindow == NULL)
{
    QUIT - There are no Protocol Windows in this Logic Analyzer
}

// Find all the images in the acquisition data.
//

// Create start and end variant parameters.
VARIANT startVar;
VARIANT endVar;
startVar.vt = VT_BSTR;
startVar.bstrVal = _T("Beginning Of Data");
endVar.vt = VY_BSTR;
endVar.bstrVal = _T("End Of Data");

// Return parameters

```



```

VARIANT_BOOL bSuccess;

BSTR bstrErrorMessage;

// Call the FindImages COM method
bSuccess = protocolWindow->FindImages(startVar, endVar,
&bstrErrorMessage);

// Check result
if (bSuccess == VARIANT_FALSE)
{
    Display bstrErrorMessage and Quit
}

```

## FindNext Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 155](#)) ]

**Applies To** • Window (see [page 136](#)) 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 154](#)).

**VB Syntax** object.FindNext

Parameters	Definition
object	An expression that evaluates to a Window (see <a href="#">page 136</a> ) object.

**Return Value** A FindResult (see [page 97](#)) object containing the results of the find.

**See Also**

- FindResult (see [page 97](#)) object
- Find (see [page 154](#)) method
- FindPrev (see [page 161](#)) method

## FindPrev Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 155](#)) ]

**Applies To** • Window (see [page 136](#)) 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 154](#)).

**VB Syntax** object.FindPrev

Parameters	Definition
object	An expression that evaluates to a Window (see <a href="#">page 136</a> ) object.

**Return Value** A FindResult (see [page 97](#)) object containing the results of the find.

- See Also**
- FindResult (see [page 97](#)) object
  - Find (see [page 154](#)) method
  - FindNext (see [page 161](#)) method

#### GetDataBySample Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 116](#)) ]

- Applies To**
- SampleBusSignalData (see [page 115](#)) 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 207](#)) 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 115</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 162</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 162](#)).

- See Also**
- GetDataByTime (see [page 164](#)) method
  - StartSample (see [page 246](#)) property
  - EndSample (see [page 232](#)) property
  - GetTime (see [page 179](#)) method

#### DataTypes and Return Values

AgtDataType	Enum Value	Return Value
AgtDataRaw	&H0001 (1)	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 85</a> ) object's ByteSize (see <a href="#">page 223</a> ) property. This can hold the maximum BusSignal size of 128 bits of unsigned data. This type is the most efficient in terms of bytes transferred. Using this type, only the smallest number of bytes needed to represent every channel in a bus/signal will be transferred. The bus/signal values are stored in the array from MSB to LSB.
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 222</a> ) property is <b>AgtBusSignalSampleNum</b> , when the GetTime (see <a href="#">page 179</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 222</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 179</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 179</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 222</a> ) property is <b>AgtBusSignalSampleNum</b> , when the GetTime (see <a href="#">page 179</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>. 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 222</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>. 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 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 207](#)) 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 115</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 162</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 162](#)).

**See Also**

- GetDataBySample (see [page 162](#)) method
- StartTime (see [page 247](#)) property
- EndTime (see [page 232](#)) property
- GetTime (see [page 179](#)) method

GetGroupCaption Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To**

- VbaViewChartData (see [page 133](#)) 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 133</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 201](#)) method

GetImageList Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**NOTE:** This method is applicable only for those Logic and Protocol Analyzer modules that support the Image View.

**Applies To** ProtocolWindow (see [page 114](#)) object

**Description** Returns a string containing the Protocol Viewer's list of images. Optionally, it writes this list to a text file.

**VB Syntax** object.**GetImageList** doWriteToFile csvFilePath success

Parameters	Definition
object	An expression that evaluates a ProtocolWindow object.

Parameters	Definition
doWriteToFile	A <b>VARIANT_BOOL</b> that specifies whether to write the image list to a file. <b>VARIANT_TRUE</b> causes a file to be written using the path in csvFilePath.
csvFilePath	A string that specifies the pathname of the optional CSV file. We ignore csvFilePath if doWriteToFile is <b>VARIANT_FALSE</b> .
success	A <b>VARIANT_BOOL</b> output parameter which returns <b>VARIANT_TRUE</b> or <b>VARIANT_FALSE</b> . <b>VARIANT_TRUE</b> means the operation was successful and the return string contains the image list. <b>VARIANT_FALSE</b> indicates an error occurred and the return string contains an error message.

**Return Value** A string containing the Protocol Viewer's list of images.

**Remark** **GetImageList** performs the same function as 'Export list to CSV...' button in the 'Image View' tab of the Protocol Viewer.

**GetImageList** returns a string. The string contains one or more lines. The first line in the string is a heading. Subsequent lines contain data. There is one data line for each image in the list. The string written to the optional file is the same as the returned string.

Each data line contains four data fields separated by commas. Here is an example of an image list containing two images.

```
Frame Number, Sample Number, Time (ns), Module
0-0, 451, 0, CSI-101
0-1, 1177, 200444481, CSI-101
```

If there are no images in the Protocol Viewer's list, then **GetImageList** method returns an error.

#### GetImageList Example

**Visual C++** **NOTE:** The example below is incomplete. The example does not contain code which acquires data. In order to work, the code in the example must have packet data in acquisition memory. It is also required that the packet data must contain some image frames.

```
// Create the Connect object.

AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));...

// Using the Connect object, obtain the IInstrument interface.

AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostName);...

// Using the IInstrument interface, obtain the IWindows interface

AgtLA::IWindowsPtr windows = pInst->GetWindows();...

// Loop through all the windows in the instrument looking for the first
window

// that presents an IProtocolWindow interface.

AgtLA::IprotoWindowPtr protocolWindow = NULL;

for (long i=0; i < windows->GetCount(); i++)
{
    // Using the IWindows interface, obtain a IWindow interface.
```

```

AgtLA::IWindowPtr window = windows->GetItem(_variant_t(i));

// Check if this window is a ProtocolWindow by type-casting the
// module pointer.
protocolWindow= windows->GetItem(_variant_t(i));
if(protocolWindow != NULL)
{
    // Found a ProtocolWindow
    break;
}
}
if (protocolWindow == NULL)
{
    QUIT - There are no Protocol Windows in this Logic Analyzer
}
// Obtain the list of images.
//
// Create parameters for optional write to text file.
VARIANT_BOOL bWriteToFile = VARIANT_FALSE;
VARIANT_BOOL bSuccess;
_bstr_t bstrPath = "";

// Return variables.
BSTR bstrReturn;

// Call the GetImageList COM method
bstrReturn = protocolWindow->GetImageList(bWriteToFile, bstrPath,
&bSuccess);

// Check result
If (bSuccess == VARIANT_FALSE)
{
    Display error message in bstrReturn and Quit
}

```

GetLine Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To** • PattgenModule (see [page 106](#)) 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 106</a> ) object.
LineNumber	A Long representing the line number from which to get the instruction or vector.

**Return Value** A String containing the InstructionOrVector (see [page 168](#)) at the specified line number.

**See Also** • SetLine (see [page 201](#)) method

#### InstructionOrVector

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To** • GetLine (see [page 167](#)) method  
• InsertLine (see [page 188](#)) method  
• SetLine (see [page 201](#)) 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_val ue>	'<parameter_name>' = <value>



<code>&lt;wait_pattern&gt;</code>	A 3-digit binary number as in the user-interface (for example, 001).
<code>&lt;module_name&gt;</code>	The name of a module in the frame.
<code>&lt;value&gt;</code>	<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 167](#)) method
- InsertLine (see [page 188](#)) method
- SetLine (see [page 201](#)) method

GetLineLabel Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

### Applies To

- PattgenModule (see [page 106](#)) 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 106</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 202](#)) method

GetModuleByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 170](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
Name	A <b>String</b> containing the module's name as defined by the Name (see <a href="#">page 238</a> ) property.

**Return Value** A hardware Module (see [page 105](#)) object with the module name given.

**See Also** • Modules (see [page 238](#)) property  
• Name (see [page 238](#)) property

GetModuleByName Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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::IAnalyzerModulePtr 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 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 207](#)) 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 115</a> ) object.
StartPosition EndPosition	<b>Variants</b> specifying the data range (see <a href="#">page 152</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.

#### GetPacketCount Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** ProtocolWindow (see [page 114](#)) object

**Description** Given a channel name, returns the number of packets for that particular channel.

**VB Syntax** object.**get\_PacketCount** samplechannel packetCount success

Parameters	Definition
object	An expression that evaluates a ProtocolWindow (see <a href="#">page 114</a> ) object.
samplechannel	A sample channel must be a channel name.
packetCount	An <b>INTEGER</b> output parameter which returns the number of packets.
success	A <b>VARIANT_BOOL</b> output parameter which returns <b>VARIANT_TRUE</b> or <b>VARIANT_FALSE</b> .

**Return Value** Number of packets for a channel

**Remarks** **PacketCount** is one of the several methods that enables the COM user to count the number of packets for a given channel.

#### GetPacketCount Example

**Visual C++** **NOTE:** The example below is incomplete. The example does not contain code which acquires data. In order to work, the code in the example must have packet data in acquisition memory. It is also required that the packet data must contain some image frames.

```

// Create the Connect object.
AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));...

// Using the Connect object, obtain the IInstrument interface.
AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostName);...

// Using the IInstrument interface, obtain the IWindows interface
AgtLA::IWindowsPtr windows = pInst->GetWindows();...

// Loop through all the windows in the instrument looking for the first
window

// that presents an IProtocolWindow interface.
AgtLA::IprotoWindowPtr protocolWindow = NULL;
for (long i=0; i < windows->GetCount(); i++)
{
    // Using the IWindows interface, obtain a IWindow interface.
    AgtLA::IWindowPtr window = windows->GetItem(_variant_t(i));

    // Check if this window is a ProtocolWindow by type-casting the
    // module pointer.
    protocolWindow= windows->GetItem(_variant_t(i));
    if(protocolWindow != NULL)
    {
        // Found a ProtocolWindow
        break;
    }
}
if (protocolWindow == NULL)
{
    QUIT - There are no Protocol Windows in this Logic Analyzer
}

// call the packet event.
// Parameter to specify a folder
_bstr_t bstrChannel= ""; // Channel name
Int packetCount // No of packet for given channel name
VARIANT_BOOL bSuccess; //

```

```
// Call the packetcount COM method
bSuccess = protocolWindow->GetPacketCount(bstrChannel, &packetCount);
// Check result
If (bSuccess == VARIANT_FALSE)
{
    Display error message in bstrErrorMessage and Quit
}
```

#### GetProbeByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
Name	A <b>String</b> containing the probe's name as defined by the Name (see <a href="#">page 238</a> ) property.

**Return Value** A Probe (see [page 111](#)) object with the name given.

**See Also**

- Probe (see [page 111](#)) object
- Probes (see [page 111](#)) object
- Instrument (see [page 98](#)) object
- Name (see [page 238](#)) property

#### GetProtocolDataFields Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • ProtocolWindow object (see [page 114](#))

**Description** Gets the acquisition data for the fields displayed in the Protocol Viewer.

**VB Syntax** object.**GetProtocolDataFields** start end channel success

Parameters	Definition
object	An expression that evaluates to a ProtocolWindow (see <a href="#">page 114</a> ) object.
start	A <b>VARIANT</b> which specifies the first packet in a range of packets for which the acquisition data is fetched. If <i>start</i> is an integer, then <i>start</i> is interpreted as the packet number for the first packet in the range. If <i>start</i> is a floating point number, then <i>start</i> is interpreted as the packet time of the first packet in the range.
end	A <b>VARIANT</b> which specifies the last packet in a range of packets for which the acquisition data is fetched. If <i>end</i> is an integer, then <i>end</i> is interpreted as the packet number for the last packet in the range. If <i>end</i> is a floating point number, then <i>end</i> is interpreted as the packet time for the last packet in the range.  <b>NOTE:</b> "start" and "end" must either be both integers (i.e. packets numbers) or be both floating point numbers (i.e. packet times).
channel	A <b>string</b> which specifies the channel (or direction) of the sample numbers specified as the start and end parameters. channel is used only if start and end are integers representing packet numbers. channel is ignored if start and end are floating point numbers representing packet times.
success	A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE. VARIANT_TRUE means the operation was successful and the return string contains data. VARIANT_FALSE means an error occurred and the return string contains an error message.

**Remark** GetProtocolDataFields returns a string that contains one or more lines. The first line in the string is a heading. Subsequent lines contain acquisition data. For each packet in the specified range of packets, there is one acquisition data line. Each data line contains a series of values separated by commas. This series of values is as per the fields displayed in the Packets pane of the Protocol Viewer. If you want to choose the fields that the method returns in a data line, you must insert, delete, or re-arrange the fields according to your requirements in the Packets pane of the Protocol Viewer.

#### GetRawData Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • AnalyzerModule (see [page 84](#)) 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 207](#)) 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 84</a> ) object.
StartPosition EndPosition	<b>Variants</b> specifying the data range (see <a href="#">page 152</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.

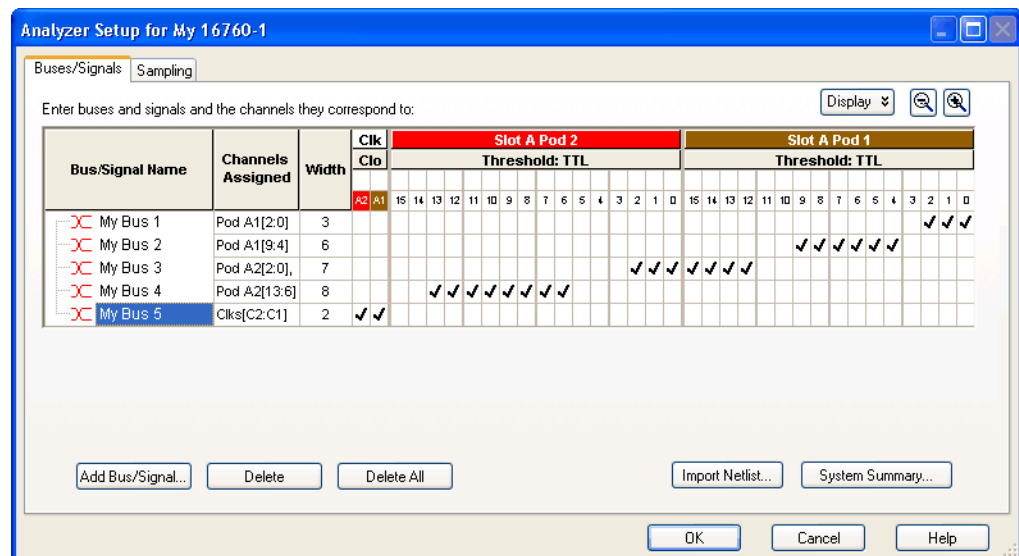
## Return Values

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 177](#)) method

[GetRawTimingZoomData](#) Method

[ [Automation Home](#) (see [page 3](#)) ] [ [Objects](#) (see [page 83](#)) ] [ [Example](#) ]

**Applies To** • [AnalyzerModule](#) (see [page 84](#)) 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 207](#)) 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 84</a> ) object.
StartPosition EndPosition	<b>Variants</b> specifying the data range (see <a href="#">page 152</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 176](#)) method

GetRemoteInfo Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Connect (see [page 95](#)) 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 243](#)) and RemoteUserName (see [page 243](#)).

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

Parameters	Definition
object	An expression that evaluates to a Connect (see <a href="#">page 95</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 243](#)) property
  - RemoteUserName (see [page 243](#)) property

#### GetSampleNumByTime Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 116](#)) ]

- Applies To**
- SampleBusSignalData (see [page 115](#)) 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 207](#)) 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 115</a> ) object.
Time	A <b>Double</b> containing the time that you want to get the sample number for. <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 83](#)) ] [ Example (see [page 116](#)) ]

- Applies To**
- SampleBusSignalData (see [page 115](#)) 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 207](#)) 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 115</a> ) object.
StartPosition EndPosition	<b>Variants</b> specifying the data range (see <a href="#">page 152</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 162</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 162](#)).

**See Also**

- GetDataByTime (see [page 164](#)) method
- GetDataBySample (see [page 162](#)) method

GetToolByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</a> ) object.
Name	A <b>String</b> containing the tool's name as defined by the Name (see <a href="#">page 238</a> ) property.

**Return Value** A Tool (see [page 129](#)) object with the tool name given.

**See Also**

- Tools (see [page 251](#)) property
- Name (see [page 238](#)) property

GetTriggerSampleNumber Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To**

- ProtocolWindow (see [page 114](#)) object

**Description** Gets the packet number and channel of the trigger packet.

**VB Syntax** object.GetTriggerSampleNumber success PacketNumber

Parameters	Definition
object	An expression that evaluates to a ProtocolWindow (see <a href="#">page 114</a> ) object.
success	A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE. VARIANT_TRUE means the operation was successful and the return string contains the channel name. VARIANT_FALSE means an error occurred and the return string contains an error message.
PacketNumber	An <b>integer</b> output parameter which returns the packet number of the trigger packet.

**See Also** • GetProtocolDataFields (see [page 175](#)) method

GetValueCaption Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • VbaViewChartData (see [page 133](#)) 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 133</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 203](#)) method

GetWindowByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
Name	A <b>String</b> containing the window's name as defined by the Name (see <a href="#">page 238</a> ) property.

**Return Value** A Window (see [page 136](#)) object with the tool name given.

**See Also** • Window (see [page 136](#)) object  
 • Windows (see [page 137](#)) object  
 • Instrument (see [page 98](#)) object  
 • Name (see [page 238](#)) property

## GoOffline Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 182](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.

**See Also** • GoOnline (see [page 186](#)) method  
• IsOnline (see [page 188](#)) 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 Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 Keysight 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/Keysight 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::IIInstrumentPtr 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 83](#)) ] [ Example (see [page 182](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 182](#)) 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 98</a> ) object.
ComputerNameOrIPAddress	A <b>String</b> containing the frame's computer host name as defined by the ComputerName (see <a href="#">page 227</a> ) property or IP address as defined by the IPAddress (see <a href="#">page 235</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 182](#)) method  
• IsOnline (see [page 188](#)) method

#### GoToPosition Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Window (see [page 136](#)) 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 136</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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 187](#)) method.

**See Also** • ImportEx (see [page 187](#)) method  
• Export (see [page 152](#)) method  
• ExportEx (see [page 153](#)) method

#### ImportEx Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 187](#)) method
- Export (see [page 152](#)) method
- ExportEx (see [page 153](#)) method

InsertLine Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To**

- PattgenModule (see [page 106](#)) 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 106</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 168</a> ) to be inserted.

**See Also**

- RemoveLine (see [page 198](#)) method

IsOnline Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 182](#)) ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</a> ) object.
ComputerName	A <b>String</b> that is initialized to the frame's name (as defined by the ComputerName (see <a href="#">page 227</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 182](#)) method
  - GoOnline (see [page 186](#)) method

## IsTimingZoom Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- BusSignal (see [page 85](#)) 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 85</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 83](#)) ] [ Example ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</a> ) object.
SaveChanges	A <b>Boolean</b> that specifies whether changes to the current configuration should be saved.
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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 200](#)) method

#### PanelLock Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 190](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 193](#)) method

#### PanelLock Example

**Visual Basic**

```
' When using Visual Basic outside of the Keysight Logic Analyzer
' application, you must create the Connect object (see
' page 95) 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 Keysight 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/Keysight 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 83](#)) ] [ Example (see [page 190](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.

**See Also** • PanelLock (see [page 190](#)) method

#### QueryCommand Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 148](#)) ]

**Applies To** • Instrument (see [page 98](#)) object  
 • Module (see [page 105](#)) object  
 • Probe (see [page 111](#)) object

- Tool (see [page 129](#)) object
- Window (see [page 136](#)) 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	A <b>String</b> that contains either an XML-based query command or an XML-based filter. If a filter is specified, the query command will be set to "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. If the 'Query' starts with the character '<', it is treated as an XML-based filter (see <a href="#">page 194</a> ); otherwise, it is treated as an XML-based query (see <a href="#">page 194</a> ) command.
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 Keysight Logic Analyzer
' application, you must create the Connect object (see
page 95) 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 83](#)) ] [ Example (see [page 27](#)) ]

**Applies To** • AnalyzerModule (see [page 84](#)) object  
• SerialModule object (see [page 128](#))

**Description** Loads a previously saved trigger file located on the instrument's file system.

**VB Syntax** object.RecallTriggerByFile TriggerFileName

Parameters	Definition
object	An expression that evaluates to an AnalyzerModule (see <a href="#">page 84</a> ) or a SerialModule (see <a href="#">page 128</a> ) object.
TriggerFileName	A <b>String</b> that contains the name of the XML-format trigger specification file located on the instrument's file system.

**Remarks** The RecallTriggerByFile method is a shortcut that reads an XML-format trigger specification file from the instrument's file system and sets the Trigger (see [page 251](#)) property of the applicable object (AnalyzerModule or SerialModule in this case).

**See Also** • AnalyzerModule (see [page 84](#)) object  
• SerialModule object (see [page 128](#))

RecallTriggerByName Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [Example]

**Applies To** • AnalyzerModule (see [page 84](#)) 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 84</a> ) object.
TriggerName	A <b>String</b> that represents the recall buffer title name.

**See Also** • AnalyzerModule (see [page 84](#)) object

## RecvFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • ConnectSystem (see [page 95](#)) object

**Description** Copies a file from the remote logic analyzer system to your local system.  
The Connect (see [page 146](#)) 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 95</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 146](#)) method

## Remove Method (BusSignals Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • BusSignals (see [page 90](#)) object

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

**VB Syntax** BusSignals (see [page 90](#)).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 83](#)) ] [ Example (see [page 101](#)) ]

**Applies To** • Markers (see [page 100](#)) 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 100</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 83](#)) ] [ Example (see [page 101](#)) ]

**Applies To** · Markers (see [page 100](#)) 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 100</a> ) object.

RemoveXML Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 101](#)) ]

**Applies To** · Markers (see [page 100](#)) object

**Description** Removes multiple Marker (see [page 100](#)) objects from the Markers (see [page 100](#)) 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 100</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 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To** · PattgenModule (see [page 106](#)) 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 106</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 188](#)) method

Reset Method (PattgenModule Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** · PattgenModule (see [page 106](#)) 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 106</a> ) object.

**See Also**

- Run (see [page 200](#)) method
- Stop (see [page 206](#)) method
- Step (see [page 205](#)) method
- Resume (see [page 199](#)) method

Resume Method (PattgenModule Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- PattgenModule (see [page 106](#)) 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 106</a> ) object.

**See Also**

- Run (see [page 200](#)) method
- Stop (see [page 206](#)) method
- Step (see [page 205](#)) method
- Reset (see [page 198](#)) method

Run Method (Instrument Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</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 206</a> ) method is called. <b>False</b> – run only once.

**See Also**

- Stop (see [page 206](#)) method
- Status (see [page 247](#)) property
- WaitComplete (see [page 207](#)) method

## Run Method (PattgenModule Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • PattgenModule (see [page 106](#)) 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 106</a> ) object.
Repetitive	A Boolean indicating the number of times the module is started. <b>True</b> – run continuously until the Stop (see <a href="#">page 206</a> ) method is called. <b>False</b> – run only once.

**See Also**

- Stop (see [page 206](#)) method
- Step (see [page 205](#)) method
- Resume (see [page 199](#)) method
- Reset (see [page 198](#)) method
- Status (see [page 247](#)) property
- WaitComplete (see [page 207](#)) method

## Save Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
SaveFileName	A <b>String</b> that contains the name of the file to be saved on the Instrument. 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 189](#)) method

## SendFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • ConnectSystem (see [page 95](#)) object



**Description** Copies a file from your local system to the remote logic analyzer system.

The Connect (see [page 146](#)) 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 95</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 146](#)) method

SetGroupCaption Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartData (see [page 133](#)) 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 133</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 165](#)) method

SetLine Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To** • PattgenModule (see [page 106](#)) 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 106</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 168</a> ) to be inserted.

**See Also** • GetLine (see [page 167](#)) method

## SetLineLabel Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 107](#)) ]

**Applies To** • PattgenModule (see [page 106](#)) 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 106</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 169](#)) method

## SetValue Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartData (see [page 133](#)) 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 133</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 202](#)) method

## SetValueArray Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • VbaViewChartData (see [page 133](#)) 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 <code>VbaViewChartData</code> (see <a href="#">page 133</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 202](#)) method

#### SetValueCaption Method

[ [Automation Home](#) (see [page 3](#)) ] [ [Objects](#) (see [page 83](#)) ] [ [Example](#) (see [page 136](#)) ]

**Applies To** • `VbaViewChartData` (see [page 133](#)) 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 <code>VbaViewChartData</code> (see <a href="#">page 133</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 181](#)) method

#### SimpleTrigger Method

[ [Automation Home](#) (see [page 3](#)) ] [ [Objects](#) (see [page 83](#)) ] [ [Example](#) (see [page 31](#)) ]

**Applies To** • `AnalyzerModule` (see [page 84](#)) 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 <code>AnalyzerModule</code> (see <a href="#">page 84</a> ) object.
OnEvent	A <b>String</b> containing the Event (see <a href="#">page 204</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 204</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 196](#)) method  
• `RecallTriggerByName` (see [page 196](#)) method

**Simple Trigger/Find Event**

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 31](#)) ]

- Applies To**
- SimpleTrigger (see [page 203](#)) method
  - Find (see [page 154](#)) method

**Description** An Event contains either a ComboValue, the String "Anything", or the String "Nothing". For the Find (see [page 154](#)) 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 154](#)) 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 203](#)) 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 154](#)) method, an Edge specification can only be one bit wide, and you cannot specify a glitch. When used with the SimpleTrigger (see [page 203](#)) 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 And DATA=eXXRXRXRX	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 203](#)) method
- Find (see [page 154](#)) method

**Step Method**

[ [Automation Home](#) (see [page 3](#)) ] [ [Objects](#) (see [page 83](#)) ] [ [Example](#) ]

**Applies To**

- PattgenModule (see [page 106](#)) 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 106</a> ) object.
Count	A <b>Long</b> indicating the number of vectors the pattern generator should output.

- See Also**
- Run (see [page 200](#)) method
  - Stop (see [page 206](#)) method

Stop Method (Instrument Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

- Applies To**
- Instrument (see [page 98](#)) 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 106](#)) object's Stop (see [page 206](#)) 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 98</a> ) object.

- See Also**
- Run (see [page 199](#)) method
  - Status (see [page 247](#)) property

Stop Method (PattgenModule Object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

- Applies To**
- PattgenModule (see [page 106](#)) 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 106</a> ) object.

- See Also**
- Step (see [page 205](#)) method
  - Resume (see [page 199](#)) method
  - Run (see [page 200](#)) method
  - Reset (see [page 198](#)) method

TestAll Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 126](#)) ]

- Applies To**
- SelfTest (see [page 126](#)) 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 126</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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
MacroName	A <b>String</b> that is the name of the VBA macro to run.

WaitComplete Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 208](#)) ]

**Applies To** • AnalyzerModule (see [page 84](#)) object  
 • Instrument (see [page 98](#)) object  
 • Module (see [page 105](#)) object

- PattgenModule (see [page 106](#)) object

**Description**

Waits until a measurement completes or a timeout (in seconds) occurs.

Executing the Instrument (see [page 98](#)) 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 105](#)) 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. <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 Keysight Logic Analyzer
    ' application, you must create the Connect object (see
    ' page 95) 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."
        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 Keysight 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/Keysight 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);
}

```

#### WriteAllImagesToFiles Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**NOTE:** This method is applicable only for those Logic and Protocol Analyzer modules that support the Image View.

**Applies To** ProtocolWindow (see [page 114](#)) object

**Description** Writes all the images in the Protocol Viewer's list to separate bitmap files in a specified folder.

**VB Syntax** Object.**WriteAllImagesToFiles** imageFolderPath errorMessage success

Parameters	Definition
object	An expression that evaluates a ProtocolWindow object.
imageFolderPath	A string that specifies the pathname of a folder where all the bitmap files are written.
errorMessage	A string output parameter. If success returns VARIANT_FALSE, then errorMessage contains information about the error that occurred. If success returns VARIANT_TRUE, then errorMessage is the empty string.
success	A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE. VARIANT_FALSE indicates an error occurred and errorMessage contains an explanation of this error.

**Remarks** **WriteAllImagesToFiles** performs the same function as the 'Save All Images...' button in the 'Image View' tab.

Each image in the Protocol Viewer's list is identified with a Frame Number string, e.g. "0-2".

**WriteAllImagesToFiles** constructs a filename for each image using the Frame Number string with the suffix ".bmp". For example, image "0-2" would be stored in a file named "0-2.bmp".

When writing files and constructing filenames, if a file with the same name already exists in the specified directory, **WriteAllImagesToFiles** overwrites the existing file with a new file.

#### WriteAllImagesToFiles Example

**Visual C++** **NOTE:** The example below is incomplete. The example does not contain code which acquires data. In order to work, the code in the example must have packet data in acquisition memory. It is also required that the packet data must contain some image frames.

```
// Create the Connect object.

AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));...

// Using the Connect object, obtain the IInstrument interface.

AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostName);...

// Using the IInstrument interface, obtain the IWindows interface

AgtLA::IWindowsPtr windows = pInst->GetWindows();...

// Loop through all the windows in the instrument looking for the first
window

// that presents an IProtocolWindow interface.

AgtLA::IprotoWindowPtr protocolWindow = NULL;
for (long i=0; i < windows->GetCount(); i++)
{
    // Using the IWindows interface, obtain a IWindow interface.

    AgtLA::IWindowPtr window = windows->GetItem(_variant_t(i));

    // Check if this window is a ProtocolWindow by type-casting the
    // module pointer.
```

```

protocolWindow= windows->GetItem(_variant_t(i));
if(protocolWindow != NULL)
{
    // Found a ProtocolWindow
    break;
}
}
if (protocolWindow == NULL)
{
    QUIT - There are no Protocol Windows in this Logic Analyzer
}

// Write All images to bitmap files in a folder
//
// Parameter to specify a folder
_bstr_t bstrPath = "."; // current working directory
BSTR bstrErrorMessage;

// Call the WriteAllImagesToFiles COM method
bSuccess = protocolWindow->WriteAllImagesToFiles(bstrPath,
&bstrErrorMessage);

// Check result
If (bSuccess == VARIANT_FALSE)
{
    Display error message in bstrErrorMessage and Quit
}

```

#### WriteImageToFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**NOTE:** This method is applicable only for those Logic and Protocol Analyzer modules that support the Image View.

**Applies To** ProtocolWindow (see [page 114](#)) object

**Description** Writes a single image to a bitmap file. The image is selected using the 'Frame Number' string from the image list.

**VB Syntax** object.**WriteImageToFile** frameNumber bitmapFilePath errorMessage success

Parameters	Definition
object	An expression that evaluates a ProtocolWindow object.
frameNumber	A string that specifies the image to be written. A Frame Number string consists of a virtual channel number and an image number separated by a dash (-). For example, a Frame Number string might be "0-1". The frame number string must exactly match the Frame Number string for one of the images in the Protocol Viewer's list of images.
bitmapFilePath	A string that specifies the pathname for the bitmap file.
errorMessage	A string output parameter. If success returns VARIANT_FALSE, then errorMessage contains information about the error that occurred. If success returns VARIANT_TRUE, then errorMessage is the empty string.
success	A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE. VARIANT_FALSE indicates an error occurred and errorMessage contains an explanation of this error.

**Remarks** **WriteImageToFile** performs the same function as the 'Show Image...' button in the 'Image View' tab followed by the 'Save Image' button in the 'Extracted Image' window.

If the file specified by bitmapFilePath already exists, **WriteImageToFile** overwrites the existing file with a new file.

#### WriteImageToFile Example

**Visual C++** **NOTE:** The example below is incomplete. The example does not contain code which acquires data. In order to work, the code in the example must have packet data in acquisition memory. It is also required that the packet data must contain some image frames.

```
// Create the Connect object.

AgtLA::IConnectPtr pConnect =
AgtLA::IConnectPtr(__uuidof(AgtLA::Connect));...

// Using the Connect object, obtain the IInstrument interface.

AgtLA::IInstrumentPtr pInst = pConnect->GetInstrument(hostName);...

// Using the IInstrument interface, obtain the IWindows interface

AgtLA::IWindowsPtr windows = pInst->GetWindows();...

// Loop through all the windows in the instrument looking for the first
window

// that presents an IProtocolWindow interface.

AgtLA::IprotoWindowPtr protocolWindow = NULL;

for (long i=0; i < windows->GetCount(); i++)
{
    // Using the IWindows interface, obtain a IWindow interface.

    AgtLA::IWindowPtr window = windows->GetItem(_variant_t(i));

    // Check if this window is a ProtocolWindow by type-casting the
    // module pointer.
```

```

protocolWindow= windows->GetItem(_variant_t(i));
if(protocolWindow != NULL)
{
    // Found a ProtocolWindow
    break;
}
}
if (protocolWindow == NULL)
{
    QUIT - There are no Protocol Windows in this Logic Analyzer
}
// Write a single image to a bitmap file
//
// Parameters to specify a Frame Number string and a bitmap file path
_bstr_t bstrFrameNumber = "0-0";
_bstr_t bstrPath = "Image.bmp";
BSTR bstrErrorMessage;
// Call the WriteImageToFile COM method
bSuccess = protocolWindow->WriteImageToFile(bstrFrameNumber, bstrPath,
&bstrErrorMessage);
// Check result
If (bSuccess == VARIANT_FALSE)
{
    Display error message in bstrErrorMessage and Quit
}

```

#### WriteOutput Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • VbaViewWindow (see [page 136](#)) object

**Description** Writes a string to the output window.

**VB Syntax** object.WriteOutput.String

Parameters	Definition
object	An expression that evaluates to a VbaViewWindow object.
String	A String to be written to the VbaView window.

#### WriteProtocolDataFieldsToFile Method

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • ProtocolWindow object (see [page 114](#))

**Description** Writes the acquisition data displayed in various fields in the Protocol Viewer to a specified CSV file.

**VB Syntax** object.WriteProtocolDataFieldsToFile start end channel csvfile success

Parameters	Definition
object	An expression that evaluates to a ProtocolWindow (see <a href="#">page 114</a> ) object.
start	<p>A <b>VARIANT</b> which specifies the first packet in a range of packets for which the acquisition data is to be written to the specified file.</p> <p>If start is an integer, then start is interpreted as a packet number. If start is a floating point number, then start is interpreted as a packet time.</p>
end	<p>A <b>VARIANT</b> which specifies the last packet in a range of packets for which the acquisition data is to be written to the specified file. If end is an integer, then end is interpreted as a packet number. If end is a floating point number, then end is interpreted as a packet time.</p> <p><b>NOTE:</b> "start" and "end" must either be both integers (i.e. packets numbers) or be both floating point numbers (i.e. packet times).</p>
channel	<p>If the value for this string is specified as <i>all</i>, then all the acquired packets data is written to the specified CSV file. <i>start</i> and <i>end</i> parameters are then ignored.</p> <p>Else, this string specifies the channel (or direction) of the sample numbers in the <i>start</i> and <i>end</i> parameters.</p> <p><i>channel</i> is used only if <i>start</i> and <i>end</i> are integers representing packet numbers. <i>channel</i> is ignored if <i>start</i> and <i>end</i> are floating point numbers representing packet times.</p>
csvfile	A string that specifies the path and name of the CSV file to which the acquired data is to be written.
success	<p>A <b>VARIANT_BOOL</b> output parameter which returns VARIANT_TRUE or VARIANT_FALSE.</p> <p>VARIANT_TRUE means the operation was successful and the return string contains data.</p> <p>VARIANT_FALSE means an error occurred and the return string contains an error message.</p>

**Remarks** WriteProtocolDataFieldsToFile writes the acquired data to a CSV file in one or more lines. The first line in the file is a heading. Subsequent lines contain acquisition data. For each packet in the specified range of packets, there is one acquisition data line. Each data line contains a series of values separated by commas. This series of values is as per the fields displayed in the Packets pane of the Protocol Viewer. If you want to choose the fields that the method writes in a data line, you must insert, delete, or re-arrange the fields according to your requirements in the Packets pane of the Protocol Viewer.



## Properties

- Activity Property (see [page 218](#))
- Axis Property (see [page 219](#))
- AxisBase Property (see [page 219](#))
- BackgroundColor Property (see [page 220](#))
- BitSize Property (see [page 221](#))
- BitSize Property (of VbaViewChartAxis object) (see [page 221](#))
- Bold Property (see [page 221](#))
- BusSignalData Property (see [page 222](#))
- BusSignalType Property (see [page 222](#))
- BusSignalDifferences Property (see [page 223](#))
- BusSignals Property (see [page 223](#))
- ByteSize Property (see [page 223](#))
- Caption Property (see [page 224](#))
- CardModels Property (see [page 224](#))
- Channels Property (see [page 224](#))
- Chart Property (see [page 225](#))
- ChartType Property (see [page 225](#))
- Color Property (see [page 226](#))
- Comments Property (see [page 226](#))
- ComputerName Property (see [page 227](#))
- Count Property (see [page 227](#))
- CreatorName Property (see [page 230](#))
- Data Property (see [page 230](#))
- DataType Property (see [page 231](#))
- Description Property (see [page 231](#))
- Differences Property (see [page 232](#))
- EndSample Property (see [page 232](#))
- EndTime Property (see [page 232](#))
- FaceName Property (see [page 233](#))
- Font Property (see [page 233](#))
- Found Property (see [page 233](#))
- Frame Property (see [page 233](#))
- Frames Property (see [page 234](#))
- HasLegend Property (see [page 234](#))
- HasTitle Property (see [page 234](#))
- Instrument Property (see [page 235](#))
- IPAddress Property (see [page 235](#))
- Item Property (see [page 235](#))
- Legend Property (see [page 236](#))
- Markers Property (see [page 237](#))
- Model Property (see [page 237](#))
- Modules Property (see [page 238](#))

- Name Property (see [page 238](#))
- NumLines Property (see [page 239](#))
- OccurrencesFound Property (see [page 239](#))
- Options Property (see [page 240](#))
- Overview Property (see [page 240](#))
- PanelLocked Property (see [page 240](#))
- Polarity Property (see [page 241](#))
- Position Property (see [page 241](#))
- Position Property (of VbaViewChartLegend object) (see [page 241](#))
- Probes Property (see [page 242](#))
- Reference Property (see [page 242](#))
- RemoteComputerName Property (see [page 243](#))
- RemoteUserName Property (see [page 243](#))
- RunningStatus Property (see [page 244](#))
- SampleDifferences Property (see [page 244](#))
- SampleNum Property (see [page 244](#))
- SelfTest Property (see [page 245](#))
- Setup Property (see [page 245](#))
- Size Property (see [page 246](#))
- Slot Property (see [page 246](#))
- StartSample Property (see [page 246](#))
- StartTime Property (see [page 247](#))
- Status Property (see [page 247](#))
- StatusMsg Property (see [page 248](#))
- SubrowFound Property (see [page 248](#))
- Symbols Property (see [page 249](#))
- TargetControlPort Property (see [page 249](#))
- TextColor Property (see [page 250](#))
- TimeFound Property (see [page 250](#))
- TimeFoundString Property (see [page 250](#))
- Title Property (see [page 251](#))
- Tools Property (see [page 251](#))
- Trigger Property (see [page 251](#))
- Type Property (see [page 252](#))
- Value Property (see [page 253](#))
- VBAVersion Property (see [page 253](#))
- VBE Property (see [page 253](#))
- Version Property (see [page 254](#))
- WebBrowser Property (see [page 254](#)) (for the VbaViewWindow object)
- WebBrowser Property (see [page 254](#)) (for the VbaViewWebBrowser object)
- Windows Property (see [page 255](#))
- \_NewEnum Property (see [page 255](#))

#### Activity Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</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 83](#)) ]

**Applies To** • VbaViewChart (see [page 132](#)) 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 132</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 133](#)) 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 83](#)) ]

**Applies To** • VbaViewChartAxis (see [page 133](#)) 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 132](#)) object's `ChartType` (see [page 225](#)) property).

**VB Syntax** `object.AxisBase [=Base]`

Parameters	Definition
<code>object</code>	An expression that evaluates to a <code>VbaViewChartAxis</code> (see <a href="#">page 133</a> ) object.
<code>Base</code>	An <code>AgtAxisBase</code> enumerated type value that can be one of the values described below.

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

<code>AgtAxisBase</code>	Enum Value	Description
<code>AgtAxisBaseBinary</code>	1	
<code>AgtAxisBaseHex</code>	2	
<code>AgtAxisBaseOctal</code>	3	
<code>AgtAxisBaseDecimal</code>	4	
<code>AgtAxisBaseSignedDecimal</code>	5	
<code>AgtAxisBaseFloatingPoint</code>	6	BitSize is ignored for this type

**See Also** • `BitSize` (see [page 221](#)) property (of `VbaViewChartAxis` object)

`BackgroundColor` Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To** • `Marker` (see [page 100](#)) 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
<code>object</code>	An expression that evaluates to a <code>Marker</code> (see <a href="#">page 100</a> ) object.
<code>Color</code>	A <code>Long</code> 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 83](#)) ] [ Example ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</a> ) object.

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

**See Also** • ByteSize (see [page 223](#)) property

#### BitSize Property (of VbaViewChartAxis)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • VbaViewChartAxis (see [page 133](#)) 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 133</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 219](#)) property

#### Bold Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartFont (see [page 134](#)) 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 134</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 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</a> ) object.

**Remarks** The BusSignalData property has the BusSignalData (see [page 86](#)) object type.

#### BusSignalType Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</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 179</a> ) method of the SampleBusSignalData (see <a href="#">page 115</a> ) object.

The Activity (see [page 218](#)), Channels (see [page 224](#)), and Polarity (see [page 241](#)) properties are only valid when the BusSignalType property is AgtBusSignalProbed.

#### BusSignalDifferences Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** • SampleDifference (see [page 125](#)) 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 125</a> ) object.

**Remarks** The BusSignalDifferences property has the BusSignalDifferences (see [page 86](#)) collection object type. Each item in the collection is a BusSignalDifference (see [page 86](#)) object.

#### BusSignals Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • Module (see [page 105](#)) object  
• Tool (see [page 129](#)) object  
• Window (see [page 136](#)) 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 90](#)) collection object type. Each item in the collection is a BusSignal (see [page 85](#)) object.

#### ByteSize Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 116](#)) ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</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 162](#)) method.

**See Also**    •    BitSize (see [page 221](#)) property

Caption Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To**    •    VbaViewChartTitle (see [page 135](#)) 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 135</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 83](#)) ] [ Example ]

**Applies To**    •    Module (see [page 105](#)) 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 105</a> ) object.

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

Channels Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**    •    BusSignal (see [page 85](#)) 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 85</a> ) object.
Channels	<p>A <b>String</b> containing <i>MultiplePodChannels</i> or the <b>String</b> "None" if no channels are assigned. <b>MultiplePodChannels</b> contains a comma separated list of <i>PodChannels</i>. <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:  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:  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 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewWindow (see [page 136](#)) object

**Description** Gets the Chart view.

**VB Syntax** object.Chart

Parameters	Definition
object	An expression that evaluates to a VbaViewWindow (see <a href="#">page 136</a> ) object.

**Remarks** The Chart property has the VbaViewChart (see [page 132](#)) object type.

#### ChartType Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChart (see [page 132](#)) 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 132</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 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartFont (see [page 134](#)) 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 134</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 83](#)) ]

**Applies To** • Marker (see [page 100](#)) 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 100</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 83](#)) ] [ Example ]

**Applies To** • Frame (see [page 97](#)) 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 97</a> ) object. The ComputerName property is the default property of the <b>Frame</b> (see <a href="#">page 97</a> ) object. Accordingly, you do not have to reference ComputerName explicitly, as shown in the syntax.

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

Count Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 227](#)) ]

**Applies To** • BusSignalDifferences (see [page 86](#)) object  
 • BusSignals (see [page 90](#)) object  
 • Frames (see [page 98](#)) object  
 • Markers (see [page 100](#)) object  
 • Modules (see [page 105](#)) object  
 • Probes (see [page 111](#)) object  
 • SampleDifferences (see [page 125](#)) object  
 • Tools (see [page 129](#)) object  
 • Windows (see [page 137](#)) 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 235](#)) 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 + vbCrLf
    myBusSignalChannels = myBusSignalChannels + "Bus/signal: " + _
        myBusSignal.Name
    myBusSignalChannels = myBusSignalChannels + ", Channels: " + _
        myBusSignal.Channels
Next
MsgBox "Bus/signal names and channels: " + vbCrLf + _
    myBusSignalChannels

```

```

Visual C++ //
// This simple Visual C++ Console application demonstrates how to
// use the Keysight 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/Keysight 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 83](#)) ] [ Example ]

**Applies To** · BusSignal (see [page 85](#)) 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 85</a> ) object.

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

Data Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChart (see [page 132](#)) 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 132</a> ) object.

**Remarks** The Data property has the VbaViewChartData (see [page 133](#)) object type.

DataType Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 115</a> ) object.

**Remarks** The DataType property can have values defined by the AgtDataType enumerated type. See DataTypes and Return Values (see [page 162](#)).

**See Also** • BusSignalType (see [page 222](#)) property

Description Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Frame (see [page 97](#)) object  
• Module (see [page 105](#)) 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 97</a> ) or Module (see <a href="#">page 105</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 83](#)) ] [ Example ]

**Applies To** • CompareWindow (see [page 95](#)) 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 95</a> ) object.

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

A comparison can be done directly by calling the Execute (see [page 151](#)) method or indirectly when its input data changes and comparisons are enabled in the user interface.

**See Also** • Execute (see [page 151](#)) method

## EndSample Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 115</a> ) object.

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

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

## EndTime Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 115</a> ) object.

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

**See Also** • StartTime (see [page 247](#)) property



## FaceName Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartFont (see [page 134](#)) 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 134</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 83](#)) ]

**Applies To** • VbaViewChartTitle (see [page 135](#)) 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 135</a> ) object.

**Remarks** The Font property has the VbaViewChartFont (see [page 134](#)) object type.

## Found Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • FindResult (see [page 97](#)) 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 97</a> ) object.

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

## Frame Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Module (see [page 105](#)) 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 105</a> ) object.

**Remarks** The Frame property has the Frame (see [page 97](#)) object type.

**See Also**

- Frame (see [page 97](#)) object
- Frames (see [page 98](#)) object

Frames Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]

**Applies To**

- Instrument (see [page 98](#)) 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 Instrument (see <a href="#">page 98</a> ) object.

**Remarks** The Frames property has the Frames (see [page 98](#)) collection object type. Each item in the collection is a Frame (see [page 97](#)) object.

HasLegend Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To**

- VbaViewChart (see [page 132](#)) object

**Description** Gets or sets if the legend is visible.

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

Parameters	Definition
object	An expression that evaluates to a VbaViewChart (see <a href="#">page 132</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 83](#)) ]

**Applies To**

- VbaViewChart (see [page 132](#)) object
- VbaViewChartAxis (see [page 133](#)) 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 83](#)) ] [ Example ]

**Applies To** • Connect (see [page 95](#)) 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 95</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>Keysight 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 21</a> )).

**Remarks** The Instrument property has the Instrument (see [page 98](#)) object type.

IPAddress Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Frame (see [page 97](#)) 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 97</a> ) object.

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

Item Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 227](#)) ]

**Applies To** • BusSignalDifferences (see [page 86](#)) object

- BusSignals (see [page 90](#)) object
- Frames (see [page 98](#)) object
- Markers (see [page 100](#)) object
- Modules (see [page 105](#)) object
- Probes (see [page 111](#)) object
- SampleDifferences (see [page 125](#)) object
- Tools (see [page 129](#)) object
- Windows (see [page 137](#)) 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 86</a> )	The BusSignalDifference's index.
BusSignals (see <a href="#">page 90</a> )	The BusSignal's name (as defined by the Name (see <a href="#">page 238</a> ) property).
Frames (see <a href="#">page 98</a> )	The Frame's computer name or IP address (as defined by the ComputerName (see <a href="#">page 227</a> ) or IPAddress (see <a href="#">page 235</a> ) properties, respectively).
Markers (see <a href="#">page 100</a> )	The Marker's name (as defined by the Name (see <a href="#">page 238</a> ) property).
Modules (see <a href="#">page 105</a> )	The Module's name (as defined by the Name (see <a href="#">page 238</a> ) property).
Probes (see <a href="#">page 111</a> )	The Probe's name (as defined by the Name (see <a href="#">page 238</a> ) property).
SampleDifferences (see <a href="#">page 125</a> )	The SampleDifference's index.
Tools (see <a href="#">page 129</a> )	The Tool's name (as defined by the Name (see <a href="#">page 238</a> ) property).
Windows (see <a href="#">page 137</a> )	The Window's name (as defined by the Name (see <a href="#">page 238</a> ) property).

**Remarks** The Item property has the appropriate type of the object in the collection (BusSignalDifference (see [page 86](#)), BusSignal (see [page 85](#)), Frame (see [page 97](#)), Marker (see [page 100](#)), Module (see [page 105](#)), Probe (see [page 111](#)), SampleDifference (see [page 125](#)), Tool (see [page 129](#)), or Window (see [page 136](#))).

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 83](#)) ]

**Applies To** · VbaViewChart (see [page 132](#)) object

**Description** Gets the chart legend.

**VB Syntax** object.Legend

Parameters	Definition
object	An expression that evaluates to a VbaViewChart (see <a href="#">page 132</a> ) object.

**Remarks** The Legend property has the VbaViewChartLegend (see [page 134](#)) object type.

Markers Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 101](#)) ]

**Applies To** · Instrument (see [page 98](#)) 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 98</a> ) object.

**Remarks** The Markers property has the Markers (see [page 100](#)) collection object type. Each item in the collection is a Marker (see [page 100](#)) 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 83](#)) ] [ Example ]

**Applies To** · Instrument (see [page 98](#)) object  
· Module (see [page 105](#)) 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 105</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 98</a> )	Gets the instrument's model number.
Module (see <a href="#">page 105</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 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 Instrument (see <a href="#">page 98</a> ) object.

**Remarks** The Modules property has the Modules (see [page 105](#)) collection object type. Each item in the collection is a Module (see [page 105](#)) object.

#### Name Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • BusSignal (see [page 85](#)) object  
• BusSignalDifference (see [page 86](#)) object  
• Marker (see [page 100](#)) object  
• Module (see [page 105](#)) object  
• Probe (see [page 111](#)) object  
• Tool (see [page 129](#)) object  
• Window (see [page 136](#)) 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 85</a> )	Gets or sets the name of the bus/signal.
BusSignalDifference (see <a href="#">page 86</a> )	Gets the bus/signal name associated with the bus/signal difference.
Marker (see <a href="#">page 100</a> )	Gets or sets the name of the marker.
Module (see <a href="#">page 105</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 111</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 129</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 136</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 83](#)) ]

**Applies To** • PattgenModule (see [page 106](#)) 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 106</a> ) object.

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

#### OccurrencesFound Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • FindResult (see [page 97](#)) 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 97</a> ) object.

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

#### Options Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** • CompareWindow (see [page 95](#)) 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 95</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 83](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.

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

#### PanelLocked Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 190](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
Message	If the font 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 83](#)) ] [ Example ]

**Applies To** • BusSignal (see [page 85](#)) 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 85</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 83](#)) ]

**Applies To** • Marker (see [page 100](#)) 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 100</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 83](#)) ]

**Applies To** · VbaViewChartLegend (see [page 134](#)) 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 134</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 83](#)) ] [ Example (see [page 111](#)) ]

**Applies To** · Instrument (see [page 98](#)) 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 98</a> ) object.

**Remarks** The Probes property has the Probes (see [page 111](#)) collection object type. Each item in the collection is a Probe (see [page 111](#)) 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 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** · BusSignalDifference (see [page 86](#)) 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 86</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 83](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.
RemoteComputerName	A String 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 83](#)) ]

**Applies To** • Instrument (see [page 98](#)) 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 98</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 83](#)) ] [ Example ]

**Applies To** • Module (see [page 105](#)) object

**Description** Gets the detailed running status of the module. Call this method when the Status (see [page 247](#)) 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 105</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 105</a> )	Running - the module is running Stopped - the module has stopped running Initializing - the module is initializing or calibrating Waiting - the module is waiting for an event SelfTest - the instrument is running SelfTest

**See Also** • Status (see [page 247](#)) property

#### SampleDifferences Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** • CompareWindow (see [page 95](#)) 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 95</a> ) object.

**Remarks** The SampleDifferences property has the SampleDifferences (see [page 125](#)) collection object type. Each item in the collection is a SampleDifference (see [page 125](#)) 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 83](#)) ] [ Example ]

**Applies To** • SampleDifference (see [page 125](#)) 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 125</a> ) object.

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

SelfTest Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 126](#)) ]

**Applies To** • Instrument (see [page 98](#)) object

**Description** Gets the SelfTest object.

**VB Syntax** object.SelfTest

Parameters	Definition
object	An expression that evaluates to a Instrument (see <a href="#">page 98</a> ) object.

**Remarks** The SelfTest property has the SelfTest (see [page 126](#)) object type.

**See Also** • SelfTest (see [page 126](#)) object

Setup Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 39](#)) ]

**Applies To** • AnalyzerModule (see [page 84](#)) 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 84</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 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To** • VbaViewChartFont (see [page 134](#)) 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 134</a> ) object.
Size	A Long 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 83](#)) ] [ Example ]

**Applies To** • Module (see [page 105](#)) object

**Description** Gets the module's slot location in the frame (see the Frame (see [page 233](#)) 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 105</a> ) object.

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

**See Also** • Frame (see [page 97](#)) object  
• Frames (see [page 98](#)) object

#### StartSample Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 115</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 232](#)) property

StartTime Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • SampleBusSignalData (see [page 115](#)) 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 115</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 232](#)) property

Status Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) object  
• Module (see [page 105](#)) 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 207](#)) 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 98</a> )	Running - at least one of the data acquisition modules, tools, or windows is running Stopped - data acquisition modules, tools, or windows have stopped running
Module (see <a href="#">page 105</a> )	Running - the module is running Stopped - the module has stopped running

**See Also**

- RunningStatus (see [page 244](#)) property

StatusMsg Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- Module (see [page 105](#)) 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 105</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 244](#)) or Status (see [page 247](#)) properties.

**See Also**

- RunningStatus (see [page 244](#)) property
- Status (see [page 247](#)) property

SubrowFound Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To**

- FindResult (see [page 97](#)) 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 97</a> ) object.

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



## Symbols Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • BusSignal (see [page 85](#)) 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 85</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 83](#)) ]**Applies To** • Frame (see [page 97](#)) 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 97</a> ) object.
Value	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: <ul style="list-style-type: none"> <li>▪ h - for hexadecimal</li> <li>▪ b - binary</li> <li>▪ o - octal</li> <li>▪ d - decimal</li> </ul> For example: "hf", "b11110000", "bxxxx1xxx".

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

## TextColor Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ]**Applies To** • Marker (see [page 100](#)) 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 100</a> ) object.
Color	A Long value that represents the marker text color.

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

Color values have the following hexadecimal form: 0x00BBGGRR. The low-order byte (RR) contains a value for the relative intensity of red; the second byte (GG) contains a value for green; and the third byte (BB) contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is &HFF. The color white is &H00FFFFFF, black is &H00000000, and red is &H000000FF.

## TimeFound Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • FindResult (see [page 97](#)) object**Description** Gets a Double containing the time (in seconds) the event occurred in the data. You can call the GetDataByTime (see [page 164](#)) 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 97</a> ) object.

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

## TimeFoundString Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • FindResult (see [page 97](#)) 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 97</a> ) object.

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

Title Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 136](#)) ]

**Applies To**

- VbaViewChart (see [page 132](#)) object
- VbaViewChartAxis (see [page 133](#)) 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 135](#)) object type.

Tools Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 129](#)) ]

**Applies To**

- Instrument (see [page 98](#)) 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 98</a> ) object.

**Remarks** The Tools property has the Tools (see [page 129](#)) collection object type. Each item in the collection is a Tool (see [page 129](#)) 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 83](#)) ] [ Example (see [page 35](#)) ]

**Applies To**

- AnalyzerModule (see [page 84](#)) 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 84</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 83](#)) ] [ Example ]

**Applies To**

- BusSignalData (see [page 86](#)) object
- Module (see [page 105](#)) object
- Tool (see [page 129](#)) object
- Window (see [page 136](#)) object

**Description** Gets the BusSignalData (see [page 86](#)), Module (see [page 105](#)), Tool (see [page 129](#)), or Window (see [page 136](#)) 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 115</a> )

Module Type	Module Object
Analyzer	AnalyzerModule (see <a href="#">page 84</a> )
ExternalScope	Module (see <a href="#">page 105</a> )
Import	Module (see <a href="#">page 105</a> )
Pattgen	PattgenModule (see <a href="#">page 106</a> )

Window Type	Window Object
Compare	CompareWindow (see <a href="#">page 95</a> )

## Value Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 87](#)) ]

**Applies To** • BusSignalDifference (see [page 86](#)) 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 86</a> ) object.

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

## VBVersion Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) object

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

**VB Syntax** object.VBVersion

Parameters	Definition
object	An expression that evaluates to an Instrument (see <a href="#">page 98</a> ) object.

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

## VBE Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]

**Applies To** • Instrument (see [page 98](#)) 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 98</a> ) object.

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

## Version Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example ]**Applies To** • Instrument (see [page 98](#)) 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 98</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 83](#)) ] [ Example (see [page 136](#)) ]**Applies To** • VbaViewWindow (see [page 136](#)) 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 136</a> ) object.

**Remarks** The WebBrowser property has the VbaViewWebBrowser (see [page 135](#)) object type.**Requirements** • Version (see [page 42](#)): 3.20 or later.

## WebBrowser Property (for VbaViewWebBrowser object)

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 135](#)) ]**Applies To** • VbaViewWebBrowser (see [page 135](#)) 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 135</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 42](#)): 3.20 or later.

## Windows Property

[ Automation Home (see [page 3](#)) ] [ Objects (see [page 83](#)) ] [ Example (see [page 137](#)) ]

**Applies To** • Instrument (see [page 98](#)) object

**Description** Gets a collection of all windows/viewers.

**VB Syntax** `object.Windows`

Parameters	Definition
<code>object</code>	An expression that evaluates to an Instrument (see <a href="#">page 98</a> ) object.

**Remarks** The Windows property has the Windows (see [page 137](#)) collection object type. Each item in the collection is a Window (see [page 136](#)) 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 83](#)) ] [ Example (see [page 256](#)) ]

**Applies To** • BusSignalDifferences (see [page 86](#)) object  
 • BusSignals (see [page 90](#)) object  
 • Frames (see [page 98](#)) object  
 • Markers (see [page 100](#)) object  
 • Modules (see [page 105](#)) object  
 • Probes (see [page 111](#)) object  
 • SampleDifferences (see [page 125](#)) object  
 • Tools (see [page 129](#)) object  
 • Windows (see [page 137](#)) object

**Description** References objects in a collection.

**VB Syntax** `object._NewEnum`

Parameters	Definition
<code>Object</code>	With Visual C++, you can browse a collection to find a particular item by using the <code>_NewEnum</code> property or the <code>Item</code> (see <a href="#">page 235</a> ) property. In Visual Basic, you do not need to use the <code>_NewEnum</code> property because it is automatically used in the implementation of <code>For Each ... Next</code> .

**Remarks** The `_NewEnum` property has the appropriate type of the object in the collection (BusSignalDifference (see [page 86](#)), BusSignal (see [page 85](#)), Frame (see [page 97](#)), Marker (see [page 100](#)), Module (see [page 105](#)), SampleDifference (see [page 125](#)), Tool (see [page 129](#)), or Window (see [page 136](#))).

With Visual C++, you can browse a collection to find a particular item by using the `_NewEnum` property or the `Item` (see [page 235](#)) 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 Keysight 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/Keysight 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

- Five new methods: "FindImages", "getImageList", "WriteImageToFile", "WriteAllImagesToFiles" and "get\_PacketCount" have been added.



# Index

## Symbols

\_NewEnum example, 256  
\_NewEnum property, 255

## A

Activity property, 218  
Add method, BusSignals object, 143  
Add method, Markers object, 143  
AddPointArrays method, 144  
AddXML method, 143  
advanced triggers, COM  
    automation, 35  
agClientSvr.tlh header file, Visual  
    C++, 51  
AgtDataType, 162  
AnalyzerModule object, 84  
automation overview, 13  
Axis property, 219  
AxisBase property, 219

## B

BackgroundColor property, 220  
BitSize property, 221  
BitSize property (of VbaViewChartAxis  
    object), 221  
Bold property, 221  
BusSignal object, 85  
BusSignalData object, 86  
BusSignalData property, 222  
BusSignalDifference object, 86  
BusSignalDifferences example, 87  
BusSignalDifferences object, 86  
BusSignalDifferences property, 223  
BusSignals example, 91  
BusSignals object, 90  
BusSignals property, 223  
BusSignalType property, 222  
ByteSize property, 223

## C

Caption property, 224  
CardModels property, 224  
changes since previous releases, 259  
Channels property, 224  
Chart property, 225  
ChartType property, 225  
Clear method, 145  
ClearOutput method, 145

Close method, 145  
Color property, 226  
COM automation, 3  
COM version checking, 42  
Comments property, 226  
CompareWindow object, 95  
ComputerName property, 227  
Connect method, 146  
Connect object, 95  
ConnectSystem object, 95  
CopyFile method, 146  
Count example, 227  
Count property, 227  
CreatorName property, 230

## D

Data property, 230  
data ranges, 152  
DataType property, 231  
DataTypes and return values, 162  
DCOM configuration, Workgroup  
    simple file sharing, 19  
DeleteFile method, 147  
Description property, 231  
Differences property, 232  
Distributed COM properties, logic  
    analyzer application, 20  
Distributed COM properties, logic  
    analyzer machine-wide, 19  
Distributed COM properties, remote  
    computer application, 21  
DoAction method, 147  
DoCommands example, 148  
DoCommands method, 147  
Draw method, 151

## E

EndSample property, 232  
EndTime property, 232  
events in Find and SimpleTrigger  
    methods, 204  
examples, LabVIEW, 54  
examples, Visual Basic programs, 27  
Excel VB macro example, 24  
Execute method, 151  
Exerciser object, 96  
Export method, 152  
ExportEx method, 153

## F

FaceName property, 233  
Find example, 155  
Find method, 154  
FindImages Method, 158  
FindNext method, 161  
FindPrev method, 161  
FindResult object, 97  
Font property, 233  
Found property, 233  
Frame object, 97  
Frame property, 233  
Frames property, 234

## G

Get/Put methods, Visual C++, 51  
GetDataBySample method, 162  
GetDataByTime method, 164  
GetGroupCaption method, 165  
GetImageList Method, 165  
GetInstrument method, Visual C++, 51  
GetLine method, 167  
GetLineLabel method, 169  
GetModuleByName example, 170  
GetModuleByName method, 170  
GetNumSamples method, 173  
GetPacketCount Method, 173  
GetProbeByName method, 175  
GetProtocolDataFields method, 175  
GetRawData method, 176  
GetRawTimingZoomData method, 177  
GetRemoteInfo method, 178  
GetSampleNumByTime method, 179  
GetTime method, 179  
GetToolByName method, 180  
GetTriggerSampleNumber  
    method, 180  
GetValueCaption method, 181  
GetWindowByName method, 181  
GoOffline example, 182  
GoOffline method, 182  
GoOnline method, 186  
GoToPosition method, 186

## H

HasLegend property, 234  
HasTitle property, 234

**I**

Import method, [187](#)  
 ImportEx method, [187](#)  
 InsertLine method, [188](#)  
 installing COM automation client software, [17](#)  
 InstructionOrVector string, [168](#)  
 Instrument object, [98](#)  
 Instrument property, [235](#)  
 IPAddress property, [235](#)  
 IsOnline method, [188](#)  
 IsTimingZoom method, [189](#)  
 Item property, [235](#)

**L**

LabVIEW, [53](#)  
 LabVIEW examples, [54](#)  
 LabVIEW tutorial, [53](#)  
 Legend property, [236](#)  
 loading configurations, COM automation, [27](#)

**M**

macro, pattern generator, [168](#)  
 macro, VBA, [24](#)  
 Marker object, [100](#)  
 Markers example, [101](#)  
 Markers object, [100](#)  
 Markers property, [237](#)  
 methods quick reference, COM automation, [70](#)  
 methods, COM automation, [141](#)  
 Model property, [237](#)  
 Module object, [105](#)  
 Modules object, [105](#)  
 Modules property, [238](#)

**N**

Name property, [238](#)  
 networking configurations supported for COM automation, [16](#)  
 New method, [189](#)  
 NumLines property, [239](#)

**O**

object hierarchy overview, [80](#)  
 object quick reference, [83](#)  
 objects quick reference, COM automation, [70](#)  
 OccurrencesFound property, [239](#)  
 Open method, [189](#)  
 Options property, [240](#)  
 Overview property, [240](#)

**P**

PanelLock example, [190](#)  
 PanelLock method, [190](#)  
 PanelLocked property, [240](#)  
 PanelUnlock method, [193](#)  
 PattgenModule example, [107](#)  
 PattgenModule object, [106](#)  
 Perl, [56](#)  
 Polarity property, [241](#)  
 Position property, [241](#)  
 Position property (of VbaViewChartLegend object), [241](#)  
 Probe object, [111](#)  
 Probes example, [111](#)  
 Probes object, [111](#)  
 Probes property, [242](#)  
 programs, Visual Basic, [27](#)  
 properties quick reference, COM automation, [70](#)  
 properties, COM automation, [217](#)  
 ProtocolWindow object, [114](#)  
 Put/Get methods, Visual C++, [51](#)  
 Python, [60](#)

**Q**

QueryCommand method, [193](#)

**R**

ranges (data), specifying, [152](#)  
 RecallTriggerByFile method, [196](#)  
 RecallTriggerByName method, [196](#)  
 RecvFile method, [197](#)  
 Reference property, [242](#)  
 reference, COM automation, [69](#)  
 RemoteComputerName property, [243](#)  
 RemoteUserName property, [243](#)  
 Remove method, BusSignals object, [197](#)  
 Remove method, Markers object, [197](#)  
 RemoveAll method, [197](#)  
 RemoveLine method, [198](#)  
 RemoveXML method, [198](#)  
 Reset method, [198](#)  
 Resume method, [199](#)  
 return values and DataTypes, [162](#)  
 Run method, [199](#)  
 Run method (PattgenModule object), [200](#)  
 running measurements, COM automation, [27](#)  
 RunningStatus property, [244](#)

**S**

sample numbers, specifying a data range by, [152](#)  
 SampleBusSignalData example, [116](#)  
 SampleBusSignalData object, [115](#)

SampleDifference object, [125](#)  
 SampleDifferences object, [125](#)  
 SampleDifferences property, [244](#)  
 SampleNum property, [244](#)  
 sampling mode, changing, [39](#)  
 Save method, [200](#)  
 SelfTest example, [126](#)  
 SelfTest object, [126](#)  
 SelfTest property, [245](#)  
 SendFile method, [200](#)  
 SerialModule object, [128](#)  
 SetGroupCaption method, [201](#)  
 SetLine method, [201](#)  
 SetLineLabel method, [202](#)  
 setting up for COM automation, [15](#)  
 Setup property, [245](#)  
 SetValue method, [202](#)  
 SetValueArray method, [202](#)  
 SetValueCaption method, [203](#)  
 simple file sharing, Workgroup DCOM configuration, [19](#)  
 simple triggers, COM automation, [31](#)  
 SimpleTrigger method, [203](#)  
 Size property, [246](#)  
 Slot property, [246](#)  
 StartSample property, [246](#)  
 StartTime property, [247](#)  
 Status property, [247](#)  
 StatusMsg property, [248](#)  
 Step method, [205](#)  
 Stop method (Instrument object), [206](#)  
 Stop method (PattgenModule object), [206](#)  
 storing captured data, COM automation, [27](#)  
 SubrowFound property, [248](#)  
 Symbols property, [249](#)

**T**

TargetControlPort property, [249](#)  
 Tcl, [64](#)  
 TestAll method, [206](#)  
 testing distributed COM connections, [18](#)  
 TextColor property, [250](#)  
 time, specifying a data range by, [152](#)  
 TimeFound property, [250](#)  
 TimeFoundString property, [250](#)  
 Title property, [251](#)  
 Tool object, [129](#)  
 Tools example, [129](#)  
 Tools object, [129](#)  
 Tools property, [251](#)  
 Trigger property, [251](#)  
 troubleshooting, Distributed COM, [19](#)  
 tutorial, LabVIEW, [53](#)  
 type library, importing, [24, 26](#)  
 Type property, [252](#)

## U

using COM automation, [23](#)

## V

Value property, [253](#)  
 VBADisplayHelpTopic method, [207](#)  
 VBARunMacro method, [207](#)  
 VBAVersion property, [253](#)  
 VbaViewChart object, [132](#)  
 VbaViewChartAxis object, [133](#)  
 VbaViewChartData object, [133](#)  
 VbaViewChartFont object, [134](#)  
 VbaViewChartLegend object, [134](#)  
 VbaViewChartTitle object, [135](#)  
 VbaViewWebBrowser example, [135](#)  
 VbaViewWebBrowser object, [135](#)  
 VbaViewWindow example, [136](#)  
 VbaViewWindow object, [136](#)  
 VBE property, [253](#)  
 version checking, COM, [42](#)  
 Version property, [254](#)  
 Visual Basic (in Visual Studio), [26](#)  
 Visual Basic examples, [50](#)  
 Visual Basic program examples, [27](#)  
 Visual C++, [51](#)

## W

WaitComplete example, [208](#)  
 WaitComplete method, [207](#)  
 WebBrowser property, [254](#)  
 what's changed, [259](#)  
 Window object, [136](#)  
 Windows example, [137](#)  
 Windows object, [137](#)  
 Windows property, [255](#)  
 Workgroup DCOM configuration,  
     simple file sharing, [19](#)  
 WriteAllImagesToFiles Method, [211](#)  
 WriteImageToFile Method, [213](#)  
 WriteOutput method, [215](#)  
 WriteProtocolDataFieldsToFile  
     method, [216](#)

