

# ABOUT AUTOMATION

This section presents an overview of the Automation capabilities of X-Stream DSOs along with an overview of the technology itself.

### CHAPTER ONE: Overview

In this chapter learn about

- *Microsoft's COM-based "OLE Automation"*
- *How to create simple applications in Visual Basic to control the instrument*
- *How to use the X-Stream Browser to interactively control the instrument*

---

## OVERVIEW OF AUTOMATION

In addition to supporting the familiar ASCII-based remote commands that have been used to control all LeCroy DSOs for many years, all of the Windows-based “X-Stream” instruments fully support control by Automation interfaces based on Microsoft’s Component Object Model (COM). Using COM, the controlling application can run directly on the instrument without requiring an external controller; or, alternatively, it can run using Microsoft’s distributed COM standard (DCOM) on a networked computer.

### Standards

Automation is a Microsoft technology, formerly referred to as “OLE Automation,” that has primarily been used to enable cross-application macro programming. It is based upon the Component Object Model (COM), which is similar in nature to CORBA, more commonly found in the UNIX world.

An application that “exposes Automation Objects” is referred to as an “Automation Server.” Automation Objects expose “Automation Interfaces” to the controlling “Automation Client.” This manual describes these Automation objects and interfaces in detail.

It is important to note that Automation itself is not language dependent; it can be used from any programming language that supports COM. This manual, however, concentrates mainly on the use of Automation from the Visual Basic Script (VBScript) language for several reasons, including the fact that it is one of the easiest to use. Also, it is the language that X-Stream instruments use for setup files (more on this later). In addition, the VBScript interpreter is installed by default on all X-Stream instruments and, therefore, is available without your having to purchase any additional software.

### Compatibility with Other LeCroy Scopes

Throughout LeCroy’s history, we have striven to maximize compatibility, and this policy remains in force. However, due to the fact that the underlying technologies used by Automation require the 32-bit Windows operating system, and that this system is available only on our X-Stream instruments, Automation is not available on the older LeCroy scope families.

### Automation and IEEE 488.2 Remote Control – How Do They Compare?

Automation does not replace the IEEE 488.2-based remote command set, which is also supported by X-Stream instruments (and will continue to be). Instead, it augments it and allows a new class of application to be created that can run on the DSO itself.

Automation however can be considered as the “Native Language,” or “Mother Tongue” of X-Stream instruments. All of the instrument’s controls and features are available to the Automation Client.

Automation controls generally are more granular than 488.2 remote commands. That is, many 488.2 remote commands set more than one control at the same time; whereas, via Automation, this is not the case.

## PART ONE: ABOUT AUTOMATION

The following table summarizes the differences between the two remote control possibilities:

|   | IEEE 488.2 Control  | Automation Control   |
|---|---|--|
| Physical Transport                                  | GPIO, TCP/IP over Ethernet  | Inter-process using COM, inter-PC using DCOM (TCP/IP)  |
| Textual parsing of instrument responses required    | Yes. All instrument responses need 'parsing' to extract useful information.                                 | No. Each element in the Automation hierarchy appears as a "variable" to the Automation client. |
| Compatibility with previous LeCroy DSOs             | Very good. In most cases remote control applications written for older DSOs will work without modification. | None. Automation is a new standard first introduced with LeCroy's X-Stream DSOs.               |
| Ability to run controlling application "in the box" | Yes, by using the TCP/IP (VICP) protocol to talk to the "local host"  | Yes, natively  |
| Ease of Use   | Not trivial. It's easier using a tool such as ActiveDSO that hides some of the complexities.                | Very easy to use from scripting languages and office productivity tools                        |
| Format of Waveform results                          | Binary or ASCII. Both require parsing before use.   | Waveforms are presented as arrays of floating point values.                                    |
| Control from MS Office suite                        | Possible via ActiveDSO utility  | Yes, natively (see examples later in this manual)  |

### General Characteristics

- When an application is running locally on the instrument and requests a connection to the DSO via Automation (for example, by using **CreateObject("LeCroy.XStreamDSO")** from Visual Basic), one of two things will happen. If the X-Stream DSO application is already running, the object returned will be a "pointer" to the running application. If the DSO application is not running, it will be started. It is not possible to run two simultaneous instances of the X-Stream DSO application.
- More than one simultaneous connection to the instrument via Automation will be accepted, but simultaneous connections are not recommended.
- When the final client has been disconnected from the instrument (server), the X-Stream DSO application will remain running and will accept further client connections.
- Operations that cause Modal Dialogs to appear in the instrument's display will, by default, disrupt access from Automation. This behavior can be changed using the **app.SystemControl.ModalDialogTimeout** and **app.SystemControl.EnableMessageBox** controls. Refer to the description of each of these controls in the reference section for more information.
- The instrument application can be minimized in order to allow the controlling application to take over the display and touch panel by means of the **app.Minimize** control. It can also be resized and repositioned on the display by means of the **app.Top**, **app.Left**, **app.Bottom**, **app.Right** controls.

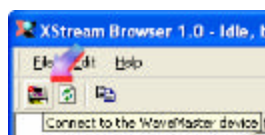
# INTRODUCTION TO THE X-STREAM BROWSER

The easiest way to get up and running with Automation, and also to visualize the “X-Stream Object Model” is to use the X-Stream browser tool, which is pre-installed on all instruments.

To launch the tool, first minimize the instrument application (**File->Minimize**), then double-click on the X-Stream Browser icon on the desktop:



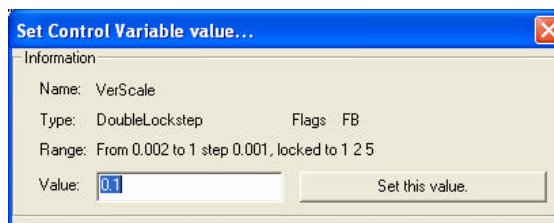
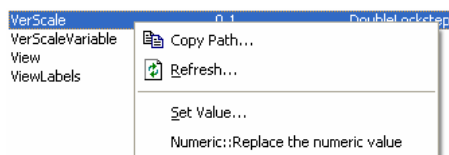
To connect to the running DSO application, click the **Connect** toolbar icon:



Upon connection, the root of the object hierarchy is shown in a layout similar to that presented when a file system is browsed using Windows Explorer.

As a quick demonstration of how the X-Stream Browser can be used, open the **Acquisition** folder, then click on the **C1** folder. Find the **VerScale** (Volts/Div) control in the right-hand window and right-click on it, then select the **Set Value** menu option.

|                              | Name                  | Value    | Type           |
|------------------------------|-----------------------|----------|----------------|
| LeCroy WaveMasterApplication |                       |          |                |
| Acquisition                  |                       |          |                |
| AuxOutput                    | AverageSweeps         | 1        | Integer        |
| C1                           | AxisRotation          | 45       | Integer        |
| C2                           | AxisYRotation         | 20       | Integer        |
| C3                           | BandwidthLimit        | Full     | Enum           |
| C4                           | ClearSweeps           |          | Action         |
| Channels                     | Coupling              | DC50     | Enum           |
| Horizontal                   | Deskew                | 0        | Double         |
| Trigger                      | InterpolateType       | Linear   | Enum           |
| WaitObject                   | Invert                | false    | Bool           |
|                              | LabelsPosition        |          | String         |
|                              | LabelsText            |          | String         |
| Cursors                      | NumSegmentsDisplayed  | 1        | Integer        |
| CustomDSO                    | Persist3DQuality      | Solid    | Enum           |
| Display                      | Persisted             | false    | Bool           |
| HardCopy                     | Persistence3d         | false    | Bool           |
| Help                         | PersistenceMonoChrome | true     | Bool           |
| Math                         | PersistenceSaturation | 50       | Integer        |
| Measure                      | PersistenceTime       | Infinite | Enum           |
| Memory                       | ProbeAttenuation      | 1        | Double         |
| PassFail                     | SegmentMode           | Adjacent | Enum           |
| Preferences                  | ShowLastTrace         | false    | Bool           |
| SaveRecall                   | StartSegment          | 1        | Integer        |
| SDA                          | UseDotJoin            | true     | Bool           |
| SystemControl                | UseGrid               | YT1      | String         |
| Utility                      | VerOffset             | 0        | Double         |
| WebEditor                    | VerScale              | 0.1      | DoubleLockstep |
|                              | VerScaleVariable      | false    | Bool           |
|                              | View                  | true     | Bool           |
|                              | ViewLabels            | false    | Bool           |



Enter a new value for **VerScale** and click the **Set this value** button. Restore the DSO's X-Stream window and note that the V/Div value for C1 should have changed to reflect the entered value.

## PART ONE: ABOUT AUTOMATION

# STEP-BY-STEP INTRODUCTION TO AUTOMATION USING VBScript

This section of the manual presents a walk-through of how to create a simple remote control application, which will run on the instrument, from scratch. It doesn't rely upon any 3<sup>rd</sup>-party development tools, since it uses Windows' built-in text editor (Notepad) and the Visual Basic Script interpreter (VBScript), which is also installed on all instruments.

1. Use the **File? Windowed** menu option to place the DSO application into windowed mode. This allows the windows start-bar to be accessed.
2. Open Windows **Notepad** via Start->Programs->Accessories->Notepad
3. Write the following text into the editor:

```
Set app = CreateObject("LeCroy.XStreamDSO")
app.AutoSetup
app.Display.GridMode = "Quad"
myVerScale = app.Acquisition.C1.VerScale
MsgBox myVerScale
```

4. Save the file to drive **D:\** and name it **Exercise1.vbs**. Leave Notepad open, we'll need it again.
5. Open Windows Explorer, via **Start? Programs? Accessories? Windows Explorer**.
6. Navigate to drive **D:\** and double-click on **Exercise1.vbs**.
7. That's it. If these steps were followed correctly, you should hear relays clicking while the scope performs an auto-setup operation and enters its quad-grid display mode.

So, what did this "program" actually do?

- The **CreateObject** statement.

```
Set app = CreateObject("LeCroy.XStreamDSO")
```

CreateObject is the Visual Basic function that creates an instance of a COM Server (a.k.a. ActiveX Control). The argument "**LeCroy.XStreamDSO**" refers to our DSO application. Once it has instantiated (connected to) our DSO application we need some kind of 'handle' (pointer) to it so that we can use it later to communicate with the instrument. **CreateObject** returns a handle to us, which we store in the **app** variable.

**NOTE:** Only a single instance of the X-Stream DSO software can run on a system at one time. If the DSO software is already running when CreateObject is called, a handle to that running instance is returned. If the DSO software is not running, it will be started.

- The **app.AutoSetup** statement.

```
app.AutoSetup
```

Using the **app** handle, this line of code calls the **AutoSetup** method, which performs the same task as the front-panel Auto-Setup button. Documentation for this method can be found later in the reference section.

- The **app.Display.GridMode = "Quad"** statement.

```
app.Display.GridMode = "Quad"
```

Using the **app** handle, this line of code sets the **GridMode** control of the **Display** system to the value "Quad". It's important to note that the controls are arranged in a hierarchy, with each 'level' of the hierarchy delimited with a decimal point ( . ).

- The **myVerScale = app.Acquisition.C1.VerScale** statement.

```
myVerScale = app.Acquisition.C1.VerScale
```

Instead of setting the value of a control, this line of code retrieves the current value of a control, in this case the Vertical Scale (Volts/Div) of Channel 1. The value returned is stored within the variable **myVerScale**.

**NOTE:** In Visual Basic Script it is not necessary to "Dimension" variables before using them (for example, using statements like "Dim myVerScale as Double").

- The **MsgBox myVerScale** statement

```
MsgBox myVerScale
```

This line of code does not communicate with the scope at all, but calls the standard Visual Basic Script function **MsgBox**. This function displays a dialog containing the value of the variable following "MsgBox". In our case the value of Channel 1's vertical scale, and waits for the **OK** button to be clicked.

Documentation about the MsgBox function can be found in Microsoft's Visual Basic Scripting documentation at [www.microsoft.com/scripting](http://www.microsoft.com/scripting).

Another point that should be mentioned here is something that is used extensively in Setup files created by the instrument: the ability to use "abbreviations" to simplify programs. Following is an example in which a shorthand method is used to replace some rather long-winded code. It is also important to note that this also enhances performance. For example, the "lookup" of the object `app.Acquisition.C1` occurs only once in the modified code, but three times in the original code.

Instead of:

```
app.Acquisition.C1.VerScale = 0.5  
app.Acquisition.C1.VerOffset = 0.1  
app.Acquisition.C1.Coupling = "DC50"
```

The following may be used:

```
set myChannell = app.Acquisition.C1  
myChannell.VerScale = 0.5  
myChannell.VerOffset = 0.1  
myChannell.Coupling = "DC50"
```

### **WHERE IS AUTOMATION USED?**

Automation is used in several places in the X-Stream based instrument.

- Instrument Setups (Panel Files)
- Custom Math/Measurements
- CustomDSO, User Interface customization
- Control from external applications (COM/DCOM)

Each of these uses is described in more detail in the following sections.



## SETUPS (PANEL FILES) ARE PROGRAMS!

Setup files, used to save and recall the state of the instrument between runs, are traditionally binary files whose internal structure is neither documented nor obvious to the user.

In XStream DSOs, however, this is no longer the case. Setups are ASCII text files that contain a complete Visual Basic Script “program” that, when “executed,” will restore the instrument to a previously recorded state. **In effect, each time a panel is saved, the instrument effectively writes you a program that, when executed, returns the instrument to the saved state.**

Due to the fact that these setups are already programs, they are a great way to get started quickly with Automation. As an example, try saving a setup into a file and examine it using a text-editor, as follows:

1. Touch **File** in the menu bar, then **Save Setup** from the drop-down menu.
2. Touch the **Browse** button and specify drive **D:** as the location to save the .Iss (LeCroy setup script) file.



3. Touch **Save Now!**
4. Minimize the application using the **Minimize** option from the **File** menu
5. Open Microsoft's Notepad application from the **Accessories** program folder (**Start? Programs? Accessories**).
6. Open the file saved above. You will see a Visual Basic Script program that begins like this:

```
' XStreamDSO ConfigurationVBScript ...
' LECROY,WM8300,WM000001,0.0.0
' Thursday, February 20, 2003 11:26:55 AM

On Error Resume Next
set XStreamDSO = CreateObject("LeCroy.XStreamDSO")
XStreamDSO.RecallingSetup = True

' AladdinPersona ...
XStreamDSO.HideClock = False
XStreamDSO.TouchScreenEnable = True
...
```

Since the entire state of the instrument, including all controls for all installed software options, is saved, this panel may look fairly complex. But don't let this fool you; the basic concept is, in fact, fairly simple.

As a quick example of how setups can be used as the starting point for controlling applications, scroll down to the end of the file and add the following code (shown in bold-type) to the file.

```
' Place any custom VBScript code after this point
'
' Perform an Auto Setup
XStreamDSO.AutoSetup
```

Add this  
code

When this setup is recalled, the complete state of the instrument will be restored, followed by an Auto-Setup operation.

Obviously this is a fairly trivial “application,” but it is easy to imagine how automated testing could be performed with the introduction of loops and conditional execution.

## PART ONE: ABOUT AUTOMATION

---

**NOTE:** Setup files stored by the instrument have file extension “.lss” (LeCroy Setup Script). These files are syntactically identical to Microsoft Visual Basic Script (VBScript) files, which have a “.vbs” extension.



**TIP:** A simple alternative to recalling the panel into the instrument is to execute it, either by double-clicking on the .lss file in Windows Explorer, or by executing it from the command line.



---

## CUSTOM MATH AND MEASUREMENTS

Custom Math and Measurements can be coded using VBScript or JavaScript in instruments equipped with the XDEV and/or XMAP software options. Using Automation control of the instrument, decisions can be made during custom processing that reconfigure the DSO.

When you are developing custom processing routines using the reference section of this manual, **app.Acquisition.Cx.Out.Result** may be used as a comprehensive reference to the **Result Object**, which is used to describe waveform data (InResult, InResult1, InResult2, OutResult).

For more detail about this capability, see the “Customization” section of the on-line Help Manual.

### CustomDSO

CustomDSO enables customization of the instrument’s UI in instruments equipped with the XDEV and/or XMAP options. Two modes of operation are supported: Basic mode and Plug-in mode.

In Basic mode a Visual Basic Script (VBScript) program can be assigned to each of 8 buttons that can, optionally, appear at the bottom of the instrument’s display. By means of Automation, each of these may further reconfigure all 8 buttons, which would allow simple menu hierarchies to be generated.

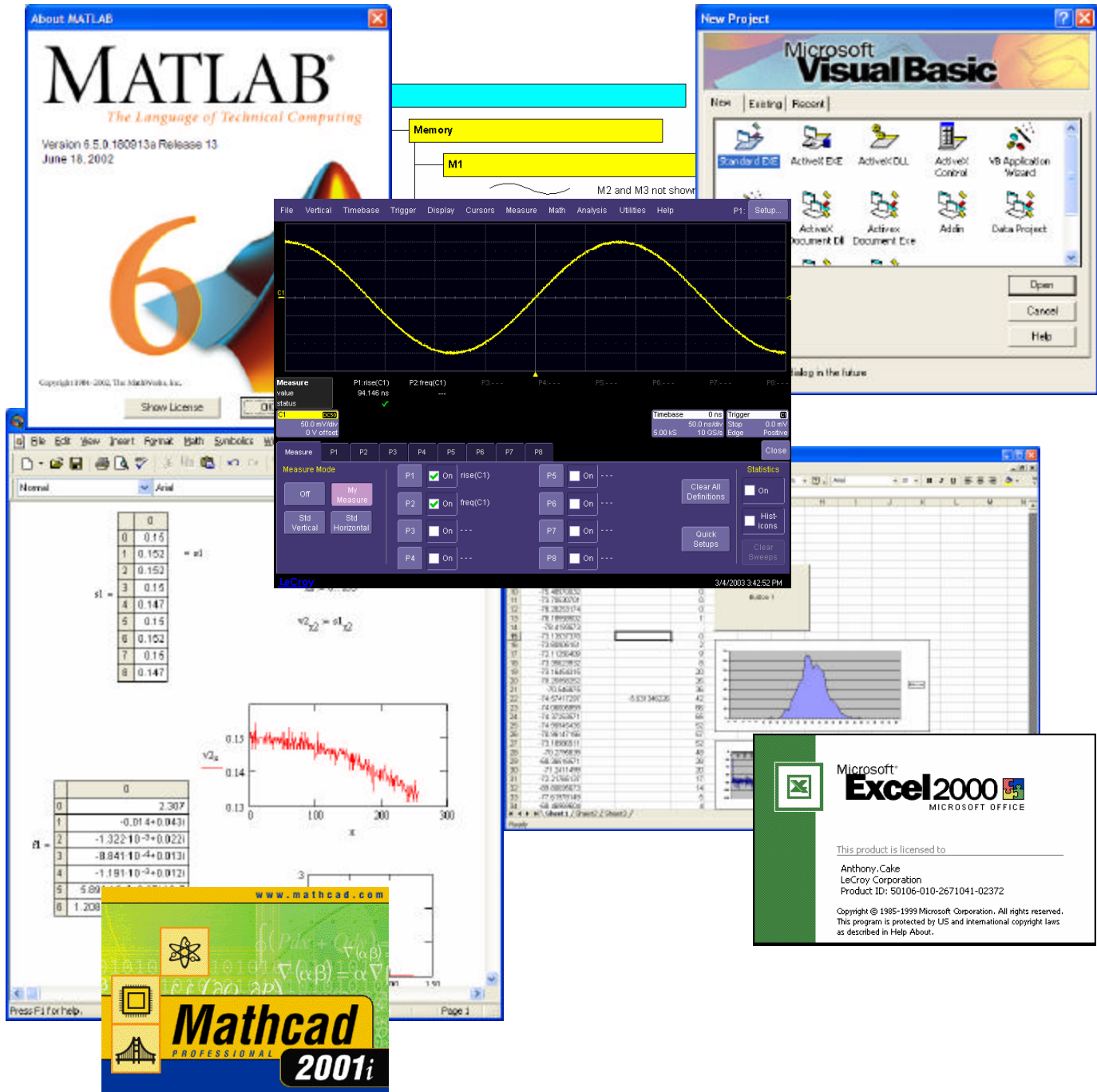
In Plug-in mode an ActiveX control, created in any of a number of programming languages, can be inserted into the instrument’s menu system. Once “embedded,” this Plug-in can take full control of the DSO using Automation.

Full documentation on CustomDSO is available in the CustomDSO section of the on-line Help Manual.

# PART ONE: ABOUT AUTOMATION

## Control from External Applications

Control of an X-Stream based instrument by Automation is possible from most modern programming languages (interpreted and/or compiled), and also from the “macro” capability of office productivity suites such as Microsoft Office.



---

**From Visual Basic**

From Visual Basic the **CreateObject** method is used to create the connection to an instrument by Automation.

The following code example creates this connection and sets up some of the instrument's controls:

```
` Connect to the X-Stream DSO
Dim app as Object
Set app = CreateObject("LeCroy.XStreamDSO")

` Setup Vertical and Horizontal settings
app.Acquisition.C1.VerScale = 0.5
app.Acquisition.C1.VerOffset = 0.25

app.Acquisition.Horizontal.HorScale = 0.000001

` Disconnect from the DSO
Set app = Nothing
```

**From MATLAB**

MATLAB uses the **actxserver** keyword to connect to the instrument.

The following code example creates this connection, enables variable vertical scale, reads the vertical scale value for C1, and increases it by a factor of 0.75.

```
DSO = actxserver('Lecroyxstreamdso')
set(DSO.Acquisition.C1.VerScaleVariable,'value',-1)
verscale = get(DSO.Acquisition.C1.VerScale,'value')
verscale = 0.75 * verscale
set(DSO.Acquisition.C1.VerScale,'value',verscale)
```

**NOTE:** Don't confuse the control of the instrument from MATLAB (MATLAB "drives") with the use of MATLAB from within a custom processing function (the instrument "drives").

# PART ONE: ABOUT AUTOMATION

---


## *From MS Office (Excel)*

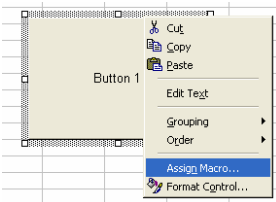
Using Automation, control of the instrument from Microsoft Excel is very similar to control from Visual Basic. This is because the “macro language” used in the office suite is Visual Basic for Applications, a lightweight version of Visual Basic.

The following example shows how to add a button to an Excel spreadsheet that connects to, and controls, the instrument on which Excel is running. Note that this example was generated using Excel 2000, other versions of Excel support similar functionality, but the specific sequence of commands may be slightly different:

1. Enable the Forms toolbar using the **View? Toolbars? Forms** menu option.



2. Click the button icon  on the toolbar and draw a button on the spreadsheet. The button will be labeled “Button 1” by default.
3. Right-click on the edge of the button and select **Assign Macro** from the drop-down menu.



4. Select the **New** button from the **Assign Macro** dialog; the macro editor will appear:



5. Type the following code into the subroutine:

```
Set app = CreateObject("LeCroy.XStreamDSO")
app.AutoSetup
Set app = Nothing
```

Clicking on the newly created button will now execute this code segment, which connects to the DSO and performs an Auto Setup.

## CONTROL BY DISTRIBUTED COM (DCOM)

Distributed COM (DCOM) can be used to control an X-Stream based instrument remotely over any network supporting TCP/IP (including the Internet).

For obvious security reasons all LeCroy instruments are shipped with this feature disabled, but the fairly simple procedure detailed below can be followed to enable DCOM and define network user accounts that have the access permissions to use it.

There are two basic ways to configure DCOM (User-Level or Share-Level), the choice of which really depends upon whether the client and server (controlling PC and instrument) are both members of the same NT Domain or not.

### Security Settings on the Instrument (Server): User Level

This mode requires that the client and server be logged into the same NT Domain. Use the share-level configuration (below) if this is not the case.

1. Run **DCOMCnfg.exe** from either **Start? Programs? Accessories? Command Prompt** or from **Start? Run...**
2. Select **LeCroyXStreamDSO** from the list of applications.
3. Click the **Properties** button.
4. Click the **General** tab and set the Authentication Level control to **Connect**.
5. Click the **Security** tab. Select **Use custom access permissions** and add the list of users that are allowed to control this scope via DCOM. Select **Everyone** (All Users) if you wish everyone to have access.
6. Click the **Identity** tab and select **The Interactive User**.

### Security Settings on the Instrument (Server): Share Level

1. Run **DCOMCnfg.exe** from either **Start? Programs? Accessories? Command Prompt** or from **Start? Run...**
2. Select **LeCroyXStreamDSO** from the list of applications.
3. Click the **Properties** button.
4. Click the **General** tab and set the Authentication Level control to **(None)**.
5. Click the **Security** tab. Select **Use custom access permissions** and add **Everyone** to the list.
6. Click the **Identity** tab and select **The interactive user**.

### Initialize the Controlling PC (Client)

1. Copy **LeCroyXStreamDSO.exe** from the **C:\Program Files\LeCroy\XStream** directory onto the controlling PC (location not important).
2. Execute **LeCroyXStreamDSO.exe**. Note that it will not run, but it will install enough information into the registry to allow remote control of other instruments.

### Connecting to a Remote Instrument

1. **Visual Basic:** Add a second argument to the **CreateObject** method, which specifies the network location of the remote instrument:

```
Set o = CreateObject("LeCroy.XStreamDSO.1", "wavemaster00121")
```

## PART ONE: ABOUT AUTOMATION

---

o.AutoSetup

2. **MATLAB:** Add a second argument to the actxserver call:

```
h = actxserver (progid [, MachineName])
```

Once connection is made using one of the above techniques, the remainder of the communication with the instrument is the same as it would be in the case where the application runs directly on the instrument.

Remember, however, that because the connection is via a network, the performance of DCOM control of an instrument will not equal that of a direct connection created by running the client application directly on the instrument.

**NOTE: Use version 2.6.0.0 or later for DCOM remote operation.**



## CONTROL VARIABLES EXPLAINED

Traditionally, properties presented to an Automation Client are simple “variables” with types such as Integer (int), String (BSTR), Floating Point (single, double), etc.

Control variables in X-Stream are an extension of the traditional Automation pattern, without affecting how they appear to most Automation clients (see section below on early/late bound clients).

As an example of what this enables, consider the following:

Take a control such as **VerScale** (Volts/Div), this may be set and queried in Visual Basic as follows:

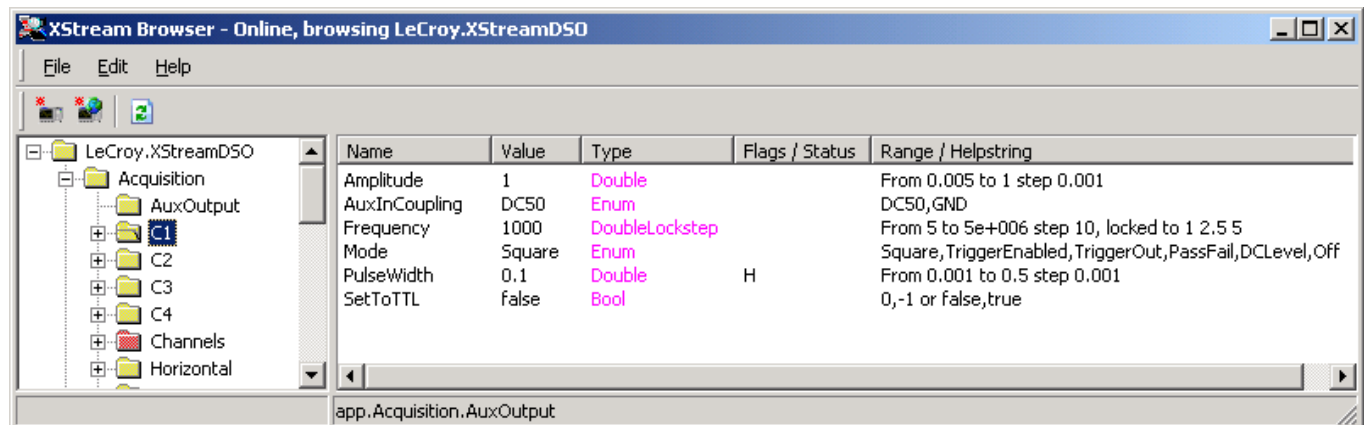
```
app.Acquisition.C1.VerScale = 0.5
CurrentValue = app.Acquisition.C1.VerScale
```

In addition, the following is supported:

```
minValue = app.Acquisition.C1.VerScale.GetMinValue
maxValue = app.Acquisition.C1.VerScale.GetMaxValue
```

This enables an Automation Client to not only set and query the current value of a control, but also to query its limits. This is useful in the generation of instrument-independent applications, or applications that present scope controls in a graphical user interface in which limits are required.

Various types of control variables are supported. The **Type** column in the X-Stream browser shows the control type:



The type designations are also given in the reference section of this manual, and are defined as follows:

|                       |  |
|-----------------------|--|
| <b>Integer</b>        | 32-bit Integer Value   |
| <b>Double</b>         | Double-precision floating point value  |
| <b>DoubleLockstep</b> | Double-precision floating point value locked to a non-linear (e.g., 1, 2, 5) sequence. |
| <b>Enum</b>           | List Value (e.g., “Orange,” “Apple,” “Pear”)   |
| <b>String</b>         | String value   |
| <b>Bool</b>           | Boolean Value { True, False }, { 0, -1 }   |
| <b>Action</b>         | Action (no arguments or value)   |

## PART ONE: ABOUT AUTOMATION

---

The properties and methods available for each control are type-specific. Listed below are the most commonly used:

| TYPE                  | PROPERTIES   |
|-----------------------|--|
| <b>Integer</b>        | VARIANT Value<br>Value(VARIANT)<br>int GetAdaptedValue<br>SetRequestedValue(int)<br>int GetRequestedValue<br>int GetDefaultValue<br>int GetGrain<br>int GetMax<br>int GetMin<br>Increment(int)   |
| <b>Double</b>         | VARIANT Value<br>Value(VARIANT)<br>double GetAdaptedValue<br>SetRequestedValue(double)<br>double GetRequestedValue<br>double GetDefaultValue<br>double GetGrainValue<br>double GetMaxValue<br>double GetMinValue<br>Increment(int)                                     |
| <b>DoubleLockstep</b> | See Double   |
| <b>Enum</b>           | VARIANT Value<br>Value(VARIANT)<br>int GetAdaptedValue<br>SetRequestedValue(int)<br>int GetRequestedValue<br>int GetDefaultValue<br>int GetMax<br>int GetMin<br>int GetNumberOfValueStates<br>Increment(int)<br>BSTR GetRangeStringScreen<br>BSTR GetRangeStringRemote |
| <b>String</b>         | VARIANT Value<br>Value(VARIANT)<br>int GetMaxLength<br>BSTR GetRequestedValue<br>BSTR GetAdaptedValue<br>SetRequestedValue(BSTR)   |
| <b>Bool</b>           | VARIANT Value<br>Value(VARIANT)<br>BOOL GetAdaptedValue<br>BOOL GetRequestedValue<br>BOOL GetDefaultValue<br>Set<br>Clear  |
| <b>Action</b>         | ActNow   |

## ACCESSING WAVEFORM/MEASUREMENT RESULTS

### Waveforms

Waveform data is exposed by a 'Result' object, which appears at various places in the object hierarchy depending upon which waveform is to be accessed. Some examples follow:

```
app.Acquisition.C1.Out.Result  
app.Math.F1.Out.Result  
app.Memory.M1.Out.Result
```

Waveform data is exposed as a simple array, no deciphering of proprietary binary formats is performed, as was necessary in the past. An example of how it is used follows.

The example is coded as an Excel macro, and should be assigned to a button as described earlier. The macro reads the number of samples in the waveform and places it in cell B1 of the Excel spreadsheet. It then reads all available sample data values and copies them into cells in the first column of the spreadsheet (A1...Axx).

```
Sub Button1_Click()  
    ' Connect to the DSO  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    ' Query the number of samples in C1 and store in cell "B1"  
    numSamples = app.Acquisition.C1.Out.Result.Samples  
    Cells(1, 2).Value = numSamples  
  
    ' Access the waveform data array, and fill the first column  
    ' of the spreadsheet with it  
    wave = app.Acquisition.C1.Out.Result.DataArray  
    For i = 0 To numSamples - 1  
        Cells(i + 1, 1).Value = wave(i)  
    Next i  
End Sub
```

**NOTE:** Ensure that the record length is < 32kSamples, since Excel has a limit on the number of rows in a spreadsheet. Ideally, you should start experimenting with short (500 point) records.

# PART ONE: ABOUT AUTOMATION

---

## Measurements

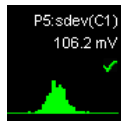
Measurement results are read in the same way as Waveforms. The following example, when copied into an Excel macro, will enable Standard Vertical parameters. It will then transfer the eight parameter values into the spreadsheet (cells C1...C8):

```
Sub Button1_Click()  
    ' Connect to the DSO  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    ' Enable Standard Vertical Parameters  
    app.Measure.MeasureMode = "StdVertical"  
  
    ' Transfer the 8 parameter values into the spreadsheet  
    Cells(1, 3).Value = app.Measure.P1.Out.Result.Value  
    Cells(2, 3).Value = app.Measure.P2.Out.Result.Value  
    Cells(3, 3).Value = app.Measure.P3.Out.Result.Value  
    Cells(4, 3).Value = app.Measure.P4.Out.Result.Value  
    Cells(5, 3).Value = app.Measure.P5.Out.Result.Value  
    Cells(6, 3).Value = app.Measure.P6.Out.Result.Value  
    Cells(7, 3).Value = app.Measure.P7.Out.Result.Value  
    Cells(8, 3).Value = app.Measure.P8.Out.Result.Value  
  
    Set app = Nothing  
End Sub
```

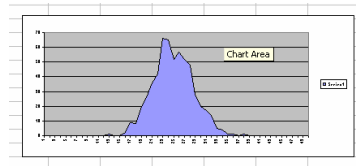
Statistics are also available for each parameter:

```
app.Measure.P1.Statistics("mean").Result  
app.Measure.P1.Statistics("max").Result  
app.Measure.P1.Statistics("min").Result  
app.Measure.P1.Statistics("num").Result  
app.Measure.P1.Statistics("sdev").Result
```

In addition, the data used to display the Histogram is available using the "histo" statistic:



```
Sub Button1_Click()  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    bins = app.Measure.P5.Statistics("histo").Result.BinPopulations  
    numBins = app.Measure.P5.Statistics("histo").Result.bins  
    For i = 0 To numBins - 1  
        Cells(i + 1, 4).Value = bins(i)  
    Next i  
  
    Set App = Nothing  
End Sub
```



## Result Status

The waveform result object described above includes a status property (bit-field) that reflects the current status of the trace. This includes both 'warning' and 'error' conditions, as described below.

| Description                      | Value              | Bit # |
|----------------------------------|--------------------|-------|
| LEC_Valid                        | 0x0                | N/A   |
| LEC_Invalid                      | 0x0000000000000001 | 0     |
| LEC_Overflow                     | 0x0000000000000002 | 1     |
| LEC_Underflow                    | 0x0000000000000004 | 2     |
| LEC_ContainsUndefinedValues      | 0x0000000000000008 | 3     |
| LEC_LessThan                     | 0x0000000000000010 | 4     |
| LEC_GreaterThan                  | 0x0000000000000020 | 5     |
| LEC_NotAPulse                    | 0x0000000000000040 | 6     |
| LEC_NotCyclic                    | 0x0000000000000080 | 7     |
| LEC_Averaged                     | 0x0000000000000100 | 8     |
| LEC_UnlockedPLL                  | 0x0000000000000200 | 9     |
| LEC_OtherError                   | 0x0000000000000400 | 10    |
| LEC_OtherWarning                 | 0x0000000000000800 | 11    |
| LEC_OtherInfo                    | 0x0000000000001000 | 12    |
| LEC_InputsIncompatible           | 0x0000100000000000 | 28    |
| LEC_AlgorithmLimitsReached       | 0x0000200000000000 | 29    |
| LEC_BadDefinition                | 0x0000400000000000 | 30    |
| LEC_TooFewData                   | 0x0000800000000000 | 31    |
| LEC_TooManyData                  | 0x0001000000000000 | 32    |
| LEC_UniformHorizIntervalRequired | 0x0002000000000000 | 33    |
| LEC_BadUnits                     | 0x0004000000000000 | 34    |
| LEC_DataRangeTooLow              | 0x0008000000000000 | 35    |
| LEC_DataUndersampled             | 0x0010000000000000 | 36    |
| LEC_PoorStatistics               | 0x0020000000000000 | 37    |
| LEC_SlowTransitionTime           | 0x0040000000000000 | 38    |
| LEC_DataResampled                | 0x0080000000000000 | 39    |
| LEC_DataInterpolated             | 0x0100000000000000 | 40    |
| LEC_MeasurementScaleImprecise    | 0x0200000000000000 | 41    |
| LEC_NoDataAvailable              | 0x0400000000000000 | 42    |
| LEC_SomeCumulatedResultsInvalid  | 0x0800000000000000 | 43    |
| LEC_InsufficientMemory           | 0x1000000000000000 | 44    |
| LEC_ChannelNotActive             | 0x2000000000000000 | 45    |
| LEC_UseStatusDescription         | 0x4000000000000000 | 46    |

## PART ONE: ABOUT AUTOMATION

---

# SYNCHRONIZATION

Synchronization or, more specifically, knowing when to read results, is critical when working with a digital oscilloscope by remote control (it is just as important by IEEE488.2 control as by Automation). This is especially true when working with an oscilloscope that uses a multithreaded architecture.

A classic problem seen in the majority of custom applications that control LeCroy (or other) DSOs is that the scope is left to free-run in Auto-trigger mode while simultaneously (and asynchronously) results are queried.

While working with the instrument via the Automation interface, there are a few techniques that can be used to guarantee the synchronization and consistency of results, whether they be waveform or parameter measurements.

The following example demonstrates a useful technique for ensuring synchronization. This example runs as an Excel macro:

```
Sub Button1_Click()  
    ' Connect to the DSO  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    ' Enable Standard Vertical parameters  
    app.Measure.MeasureMode = "StdVertical"  
  
    ' Stop the free-running trigger and take a single acquisition  
    ' Use a 10 second timeout in the case that a trigger is not detected  
    app.Acquisition.TriggerMode = "Stopped"  
    app.Acquisition.Acquire (10, True)  
  
    ' Read the first parameter value and transfer into the spreadsheet  
    Cells(1, 3).Value = app.Measure.Pl.Out.Result.Value  
  
End Sub
```

The **Acquire** method arms the acquisition system and waits for a user-specified time for a trigger. The second argument, a Boolean, specifies whether or not to force a trigger before returning if a trigger doesn't arrive within the allotted time period. The method also returns a Boolean value signifying whether or not a trigger arrived. See the reference section for more information on this useful method.

Another scenario where synchronization is necessary is between changing settings and reading results, even when no acquisition took place. For this the **WaitUntillIdle** method is used. This method is "blocking" and will not return control until the setup request has completed.

*Note that the **Acquire** method is equivalent to setting the Trigger Mode to "Single", then executing **WaitUntillIdle**.*

An example of **WaitUntilIdle** usage follows:

```
Sub Button1_Click()  
    ' Connect to the DSO  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    ' Enable Standard Vertical parameters  
    app.Measure.MeasureMode = "StdVertical"  
  
    ' Wait for the change to take place for a max. of 5 seconds  
    app.WaitUntilIdle(5)  
  
    ' Read the value of measurement P1 (pkpk) and transfer into the spreadsheet  
    Cells(1, 2).Value = app.Measure.P1.Out.Result.Value  
  
    ' Enable Standard Horizontal parameters  
    app.Measure.MeasureMode = "StdHorizontal"  
  
    ' Wait for the change to take place for a max. of 5 seconds  
    app.WaitUntilIdle(5)  
  
    ' Read the value of measurement P1 (rise time) and transfer into the  
    spreadsheet  
    Cells(1, 3).Value = app.Measure.P1.Out.Result.Value  
End Sub
```

**NOTE:** In almost all remote control applications, it is **HIGHLY RECOMMENDED** that you **STOP** acquisitions before accessing result data. Most remote control problems are caused by failure to follow this practice.

## PART ONE: ABOUT AUTOMATION

---

### GOOD PRACTICES

- Using the **app.SetToDefaultSetup** action, restore the instrument to its default state before setting the controls required by an application. This eliminates any dependency on the previous configuration of the instrument. LeCroy strives to ensure that the default state of the instrument is constant from one software release to the next.
- Synchronization is an important concept that needs to be understood before you work with an X-Stream DSO via Automation. Attempting to read results while acquisitions are in progress could cause inconsistent results.
- Use the X-Stream Browser while developing Automation applications. This tool is guaranteed to show the up-to-date status of the Automation hierarchy since it retrieves it from a running instrument. It is also a very quick and easy way to exercise controls in real-time without your having to write a single line of code.
- When using a result object, verify that the status is valid to ensure that the acquisition and/or processing was valid.

### EXAMPLES

Following are fairly complete examples of automating an X-Stream DSO, including configuration, acquisition, and reading results. Examples are given both as Excel macros, and as Visual Basic Scripts, which can run without Excel being loaded on the instrument.

#### Example 1: Excel Macro to Perform FFT of C1

```
Sub Button1_Click()  
    ' Connect to the DSO  
    Set app = CreateObject("LeCroy.XStreamDSO")  
  
    ' Restore the instrument to its default state  
    app.SetToDefaultSetup  
  
    ' Stop acquisitions during setup  
    app.Acquisition.TriggerMode = "Stopped"  
  
    ' Turn C2 off (default state leaves C1 and C2 On)  
    app.Acquisition.C2.View = False  
  
    ' Configure F1=FFT(C1), using a Blackman-Harris filter  
    app.Math.F1.View = True  
    app.Math.F1.Source1 = "C1"  
    app.Math.F1.Operator1 = "FFT"  
    app.Math.F1.Operator1Setup.Window = "BlackmanHarris"  
  
    ' Take a single acquisition, force after 2 seconds if it doesn't trigger  
    app.Acquisition.Acquire 2, True  
  
    ' Read out the FFT  
    ' Query the number of samples in F1 and store in cell "B1"  
    numSamples = app.Math.F1.Out.Result.Samples  
    Cells(1, 2).Value = numSamples  
  
    ' Access the waveform data array, and fill the first column  
    ' of the spreadsheet with it  
    wave = app.Math.F1.Out.Result.DataArray  
    For i = 0 To numSamples - 1
```



```
Cells(i + 1, 1).Value = wave(i)
Next
```

```
End Sub
```

## Example 2: VBScript Program to Perform FFT of C1 and Store Results in Text File

This example requires no additional software to be installed on the instrument, since it relies upon the built-in Visual Basic Script interpreter. The example is very similar to the previous Excel example, the most notable difference being the use of a standard system ActiveX control, “**Scripting.FileSystemObject**”, to enable the creation of files containing waveform data in ASCII format.

```
' VBScript example
' Configure the DSO to perform an FFT on Channel 1 and store
' the resulting data in a text file in ASCII format

' Connect to the DSO
Set app = CreateObject("LeCroy.XStreamDSO")

' Restore the instrument to its default state
app.SetToDefaultSetup

' Stop acquisitions during setup
app.Acquisition.TriggerMode = "Stopped"

' Turn C2 off (default state leaves C1 and C2 On)
app.Acquisition.C2.View = False

' Configure F1=FFT(C1), using a Blackman-Harris filter
app.Math.F1.View = True
app.Math.F1.Source1 = "C1"
app.Math.F1.Operator1 = "FFT"
app.Math.F1.Operator1Setup.Window = "BlackmanHarris"

' Take a single acquisition, force after 2 seconds if it doesn't trigger
app.Acquisition.Acquire 2, True

' Readout the FFT
numSamples = app.Math.F1.Out.Result.Samples

Set fso = CreateObject("Scripting.FileSystemObject")
Set MyFile= fso.CreateTextFile("c:\XStreamFFT.txt", True)

' Write the FFT power spectrum into the file, sample by sample
wave = app.Math.F1.Out.Result.DataArray
For i = 0 To numSamples - 1
    MyFile.WriteLine(wave(i))
Next

' Clean up
MyFile.Close
Set fso = Nothing
Set app = Nothing
```

## PART ONE: ABOUT AUTOMATION

---

### Example 3: Script to Measure the Rise Time of the Signal on C1 and Display It in a Popup Window

This example configures the DSO to measure the rise time of the signal on C2, take a single acquisition, and present the results in a popup dialog. The example requires no additional software to be installed on the instrument, since it relies on the built-in Visual Basic Script interpreter.

```
' VBScript example
' Configure the DSO to measure the rise time of the signal
' on Channel 1 and display it in a popup message box.

' Connect to the DSO
Set app = CreateObject("LeCroy.XStreamDSO")

' Restore the instrument to its default state
app.SetToDefaultSetup

' Stop acquisitions during setup
app.Acquisition.TriggerMode = "Stopped"

' Turn C2 off (default state leaves C1 and C2 On)
app.Acquisition.C2.View = False

' Configure Pl=rise(C1)
app.Measure.MeasureMode = "MyMeasure"
app.Measure.Pl.View = True
app.Measure.Pl.ParamEngine = "rise"

' Take a single acquisition, force after 2 seconds if it doesn't trigger
app.Acquisition.Acquire 2, True

' Present the rise time in a popup message box
MsgBox app.Measure.Pl.Out.Result.Value & "s", vbOKOnly, "Rise time of C1"

' Clean up
Set app = Nothing
```

## EARLY AND LATE BINDING

The COM standard on which Automation is built supports two kinds of “binding” between client and server: early (static), and late (dynamic, dispatch). Static binding usually involves a type library and is used primarily by compiled languages such as C++. In this case, function entry points are resolved at compile time. Dynamic binding (also known as late binding) involves resolving method and property calls at run time, as opposed to compile time.

The Automation interfaces in X-Stream based DSOs use primarily the latter: Dynamic binding. From most programming languages (VB, VBScript, etc.) this is transparent. But when you are developing applications in C++, which doesn’t provide late-binding natively, the use of a “helper” class is required. This is demonstrated below:

```
#include "stdafx.h"

#include "AtlBase.h"
CComModule _Module;
#include "AtlCom.h"

CComPtr<IDispatch> spDso;
CComDispatchDriver ddDso;      // dispatch ptr. to root of object model (app)

int main(int argc, char* argv[])
{
    printf("Hello X-Stream World!\n");

    ::CoInitialize(NULL);

    HRESULT hr = spDso.CoCreateInstance(L"LeCroy.XStreamDSO");
    if(SUCCEEDED(hr))
    {
        ddDso = spDso;

        // perform an Auto-Setup (app.AutoSetup)
        hr = ddDso.Invoke0(L"AutoSetup");

        // retrieve a Dispatch ptr. to the app.Display object
        CComVariant displayPtr;
        hr = ddDso.GetPropertyByName(L"Display", &displayPtr);
        CComDispatchDriver ddDisplay(displayPtr.pdispVal);

        // enter Dual-grid mode (app.Display.GridMode = "Dual")
        hr = ddDisplay.PutPropertyByName(L"GridMode", &CComVariant("Dual"));
    }

    return 0;
}
```

## PART ONE: ABOUT AUTOMATION

---

### VBS REMOTE COMMAND

For users who wish to harness the power of Automation control of an instrument, but are currently using “traditional” remote commands via GPIB or the network (using the VICP protocol), there is a solution. This is primarily of interest in controlling the advanced features of X-Stream DSOs, which are not supported by a traditional remote command.

X-Stream instruments, in addition to supporting LeCroy’s standard remote command set, also support a new command/query called **VBS[?]**. This command may be used in traditional remote control applications to access Automation controls.

This example shows two methods for setting the V/Div of Channel 1, the former using a traditional remote command, **VDIV**, and the latter using an Automation control via the new remote command, **VBS**. These two commands are equivalent:

```
C1:VDIV 0.5
```

```
VBS `app.Acquisition.C1.VerScale = 0.5`
```

In its query form the following are equivalent:

```
C1:VDIV?
```

```
VBS? `return = app.Acquisition.C1.VerScale`
```

A couple of points to note here are that the **app** variable is pre-defined and refers to the root of the Automation hierarchy. Also note that for the query form the **return =** is important, it indicates which value you wish to return to the caller.

The VBS[?] Command/Query is documented in more detail in the *Remote Control Manual* for X-Stream DSOs.

# X-STREAM DSO OBJECTS

The object hierarchy exposed by X-Stream based instruments is rooted at the Application object. This is the object returned when the **CreateObject("LeCroy.XStreamDSO")** method is executed in Visual Basic. All major instrument subsystems are available from this object, and many of these subsystems themselves are broken down further. Note that to simplify this figure only the first and last of the collections of Channels, Memories, Math, and Measurements are shown.

The reference section of this manual describes the controls presented by each of these objects.



**NOTE:** The root of the object hierarchies of some software options are not shown in this diagram.

**X-STREAM**