

X-Stream COM Object Programming with MATLAB®

Understanding the MATLAB Syntax for Interfacing with the LeCroy X-Stream COM Object Server

Introduction	2
About MATLAB.....	2
The LeCroy Oscilloscope COM Object Server.....	2
XStream Browser	3
Connecting to the Oscilloscope Application	4
Recommended Coding Methodology	4
Frequently Used Code Blocks	5
Code Creation Recipes	6
Additional Tips and Tricks	9
Conclusion.....	10

Introduction

Beginning with MATLAB R14, the syntax required to access properties in a hierarchical object changed significantly. For example in R13, the code to read back the time per point was the following:

```
get(app.Acquisition.Horizontal.TimePerPoint,'Value') % <= R13
```

Beginning with R14, the code is now:

```
get(app.Object.Item('Acquisition').Object.Item('Horizontal').Item('TimePerPoint'),'Value') % >=R14
```

This syntax to communicate with a hierarchical object can be difficult to get right, and the applicability of the Object.Item and Item constructs depend on the property being referenced. This document provides a higher-level understanding of the interface so that users can write the code necessary to access any scope property – without “guesswork.”

About MATLAB

MATLAB® is an interactive software environment and programming language from The MathWorks used to control instruments, make measurements, analyze data, create GUI-based applications, and build test systems. MATLAB provides interactive graphical tools and command-line functions for data analysis tasks such as digital filtering, signal processing, signal modulation, curve fitting, and statistical analysis.

MATLAB itself supports communication with instruments (including LeCroy oscilloscopes) using its Instrument Control Toolbox. The toolbox supports TCP/IP communication (whether you install MATLAB on a remote PC or directly on the LeCroy oscilloscope), MATLAB instrument drivers, and a graphical user interface (GUI) called Test & Measurement Tool that allows to communicate with the instrument without writing MATLAB code. How to use MATLAB in this way is outside the scope of this application brief. However, you can download a Getting Started Guide called “Using MATLAB with LeCroy Oscilloscopes in 15 Minutes” at www.mathworks.com/lecroy.

The LeCroy Oscilloscope COM Object Server

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” oscilloscopes 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.

The oscilloscope application is a COM object server, with an object structure that is composed of many systems, including **Acquisition**, **Math**, and **Measure**, to name just a few. (Such a server is sometimes referred to as an automation server.) Each system can include subsystems; for example, the

Acquisition system includes the **C1, C2, C3, C4, Horizontal** and **Trigger** subsystems. The result is a multi-tiered hierarchy that requires users to “drill” into the hierarchy to get to the desired property, method or action.

XStream Browser

LeCroy provides an object browser called **XStream Browser** that is pre-installed on all its Windows-based oscilloscopes. You can find a link to it on the desktop of your LeCroy scope. Use this application to determine where to find the objects, properties, actions and methods required by your application. It can also be used indirectly to determine the syntax for your MATLAB code.

Figure 1 shows the XStream Browser application. Its user interface has a look that is similar to Windows Explorer; the left-hand frame of the window uses a “tree” control to let users navigate through the object hierarchy, and the right-hand frame shows the items.

Note in figure 1 the “folders” in the left-hand frame. The systems represented by yellow folders can be considered as objects, and may contain subfolders, which are child objects. Purple folders (typically called **Result**) are objects, but are handled in a slightly different manner. Result folders contain your measurement and waveform results. Pink folders are “collections” and are not addressed in this version of the document.

Most items in the right-hand frame (except within **Result** folders) are **control variables**, or **CVARs** for short. CVARs provide an interface for accessing scope configurations and for executing methods and actions. When viewed from XStream Browser, many CVARs appear to be properties, but are actually objects with properties such as **Value, Name, Type** and **Range**, to name a few. The right-hand frame includes columns for several of the more useful properties of each CVAR.

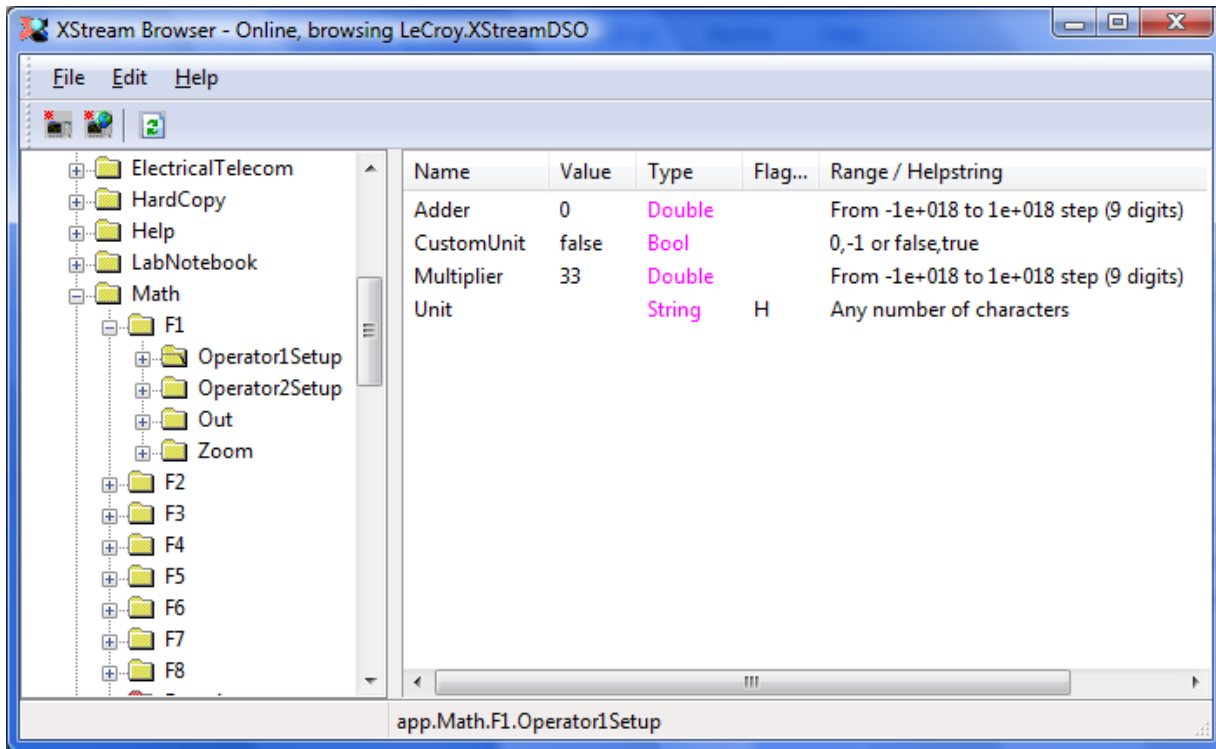


Figure 1 – View of XStream Browser. The left-hand frame shows the scope application object hierarchy; the right-hand frame shows control variables for the selected left-hand frame object.

Connecting to the Oscilloscope Application

To make a connection to the oscilloscope application, use MATLAB's `actxserver` command:

```
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')
```

The first argument is the ProgID; it always should be set to `'LeCroy.XStreamDSO.1'`. The second argument is the scope's IP address. When running MATLAB on the scope, use the IP address 127.0.0.1, which is the self-referencing IP address. Note that when running MATLAB from a remote PC, you need to configure Windows to allow DCOM connections on both the remote PC and on the oscilloscope. (See the DCOM setup in the Automation Manual for detailed instructions.)

Recommended Coding Methodology

Given the complexity of MATLAB's syntax for handling multi-tiered automation objects, it is highly recommended to create **object variables** at each level of the hierarchy. In MATLAB, this kind of variable is referred to as a **handle**. This simplifies the code, makes it more readable and easier to debug. Here are some examples of object variables that can be directly used in your script for

referencing **app.Acquisition.C1.Out.Result**, **app.Measure.P1.Out.Result** and **app.Math.F1.Out.Result**.

```
% Instantiate of Scope Application object at the IP address referenced.
% (Use 127.0.0.1 when running MATLAB on the scope)
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')

% creation of object variables 1 level down from top-level
acq = app.Object.Item('Acquisition');
math = app.Object.Item('Math');
meas = app.Object.Item('Measure');
PF = app.Object.Item('PassFail');

% creation of object variables 1 level further
c1 = acq.Object.Item('C1');
f1 = math.Object.Item('F1');
p1 = meas.Object.Item('P1');
p1Operator = p1.Object.Item('Operator')

% creation of object variables to results
c1_results = c1.Out.Result;
f1_results = f1.Out.Result;
p1_results = p1.Out.Result;
```

When using object variables, the code to access results and to get or set properties becomes much easier to read and debug:

```
% Set some properties for P1
% (assumes P1=per@level or other parameter with similar CVARS)
set(p1Operator, 'LevelType', 'Percent');
set(p1Operator, 'PercentLevel', 66);
set(p1Operator, 'Slope', 'Neg');

% Read back and display some results
p1_Val = p1_results.Value
p1_status = p1_results.Status
p1_time = p1_results.FirstEventTime
```

The following code is equivalent, but doesn't use intermediate object variables. (Note the smaller font size!)

```
% Equivalent code without use of object variables
set(app.Object.Item('Measure').Object.Item('P1').Object.Item('Operator'), 'LevelType', 'Percent')
set(app.Object.Item('Measure').Object.Item('P1').Object.Item('Operator'), 'PercentLevel', 66)
set(app.Object.Item('Measure').Object.Item('P1').Object.Item('Operator'), 'Slope', 'Neg')
val = get(app.Object.Item('Measure').Object.Item('P1').Out.Result, 'Value');
status = get(app.Object.Item('Measure').Object.Item('P1').Out.Result, 'Status');
evtime = get(app.Object.Item('Measure').Object.Item('P1').Out.Result, 'FirstEventTime');
```

Frequently Used Code Blocks

This section includes blocks of code that are frequently used by customers:

A) Accessing channel results

```
% drill into hierarchy to C1 results
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')
acq = app.Object.Item('Acquisition');
c1 = acq.Object.Item('C1');
c1_results = c1.Out.Result;

% Get entire data array
c1_data = get(c1_results, 'DataArray', -1, -1, 0, 1);

% Get C1 properties
c1_AcqTStamp = c1_results.FirstEventTime;
c1_status = c1_results.Status;
c1_NumSweeps = c1_results.Sweeps;
c1_VperStep = c1_results.VerticalPerStep;
```

B) Accessing Parameter results

```
% Drill into hierarchy to P1 results
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')
meas = app.Object.Item('Measure');
p1 = meas.Object.Item('P1');
p1_results = p1.Out.Result;

% Get entire value array
P1_array = get(p1_results, 'ValueArray', -1, 0, 1)

% Get last measured value and other properties
p1_Val = p1_results.Value
p1_status = p1_results.Status
p1_statdesc = p1_results.StatusDescription
p1_res = p1_results.VerticalResolution
p1_time = p1_results.FirstEventTime
```

Code Creation Recipes

1) Recipe for creating code that access CVARS that are not Results or Collections

This recipe applies to control variables that are in the right-hand frame when selecting an object that is represented by a yellow folder in XStream Browser.

- A) Find the control variable in XStream Browser. For example, the **VerScale** CVAR is used to program the volts per division for the input channels. The path to this is **app.Acquisition.C1.VerScale**
- B) For each object in the hierarchy, use the Object.Item('Object Name') syntax to create an object variable for that level of the hierarchy. For example, the following lines of code drill into the hierarchy to the "folder" containing the **VerScale** CVAR.

```
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')
```

```
acq = app.Object.Item('Acquisition');
c1 = acq.Object.Item('C1');
```

Note that lines 2 and 3 use the object variable created in the line before it. The result is that you have a set of object variables to use rather than the long and clumsy line of code such as the one at the beginning of the document.

- C) For control variables with values that you can read or write, use the **get** and **set** functions to access properties of the control variable. The **Value** property is most frequently used, but others are available, including **Range**.

Here are examples of getting and setting properties of the **VerScale** CVAR for channel 1:

```
% read the value of the VerScale CVAR.
C1VDiv = get(c1.Item('VerScale'), 'Value')

% read the Range and Type properties of the VerScale CVAR
range = get(c1.Item('VerScale'), 'Range')
type = get(c1.Item('VerScale'), 'Type')

% set VerScale to 0.789 V/div
set(c1.Item('VerScale'), 'Value', 0.789)
```

- D) For control variables that are Actions, use the **invoke** function. Action CVARs have an interface called **ActNow** that can be referenced, but this isn't required.

```
% execute an application-level Clear Sweeps action
app.invoke('ClearSweeps', 'ActNow');

% execute an AutoSetup action
app.invoke('AutoSetup', 'ActNow');

% execute the ResetAll action for the Math system
math.invoke('ResetAll', 'ActNow');
```

- E) For control variables that are Methods, use the **invoke** function, and include the arguments required to perform the method (see the Automation manual for details)

```
% execute the Acquire method with 5s timeout, without forcing a trigger
acq.invoke('Acquire', 5, false);
```

2) Recipe for accessing items within Result Objects

This recipe applies to control variables that are in the right-hand frame when selecting an object that is represented by a purple folder in XStream Browser.

- A) Find the control variable in XStream Browser. For example, the **Value** property for P1 is available when a measurement parameter P1 is configured and turned on. The path to this is **app.Measure.P1.Out.Result.Value**. Note that Value is a property rather than a CVAR; this is

the biggest difference between items under a Results folder rather than a yellow folder like the Acquisition object, which has CVARs listed in the right-hand pane when selected.

- B) For each object in the hierarchy up to **but not including the Result object's parent** object, use the Object.Item('Object Name') syntax to create an object variable for that level of the hierarchy.

For example, the following lines of code drill into the hierarchy to the **P1** object, which is the **Result** object's parent object's ("**Out**") parent.

```
app = actxserver('LeCroy.XStreamDSO.1', '172.28.15.55')
meas = app.Object.Item('Measure');
p1 = meas.Object.Item('P1');
```

- C) Create an object variable for the **Result** object using this syntax:

```
p1_results = p1.Out.Result
```

The measurement system provides other Result interfaces as well. Here are some additional examples:

```
p1minResult = p1.min.Result
p1maxResult = p1.min.Result
```

- D) To read properties within the Result object, use one of the two syntaxes:

```
p1_val = p1_results.Value
        or
p1 = get(p1_results, 'Value')
```

Clearly the first syntax is easier on the eyes, especially when you are returning multiple **Result** properties:

- E) To read back multi-valued data such as the **DataArray** and **ValueArray** properties use the **get** function. **ValueArray**, which returns all measured values for a parameter, and **DataArray**, which returns the data for a waveform. **DataArray** and **ValueArray** are actually interfaces that require several arguments that determine the number of values to return what kind of values, etc. See the Automation Manual (rev B or higher) for information on accessing **ValueArray** and **DataArray**.

```
P1_array = get(p1_results, 'ValueArray', -1, 0, 1)
C1_data = get(c1_results, 'DataArray', -1, -1, 0, 1)
```

Additional Tips and Tricks

In MATLAB, a semicolon suppresses the output that is otherwise generated. Leave off the semicolon when developing your code in order to glean some insight about the oscilloscope object structure.

Examples:

A) Creation of object variable to the Acquisition object

```
>> acq = app.Object.Item('Acquisition')
acq =
    Interface.LeCroy_Maui_Persona_Interfaces_1.0_Type_Library.ILecMauiAcquisitionAutomation
```

B) Creation of object variable to a parameter **Result** object

```
>> P1OutResult = P1.Out.Result
P1OutResult =
    Interface.LeCroy_ResultAutomationInterfaces_1.0_Type_Library.ILecAutomationParamResult
```

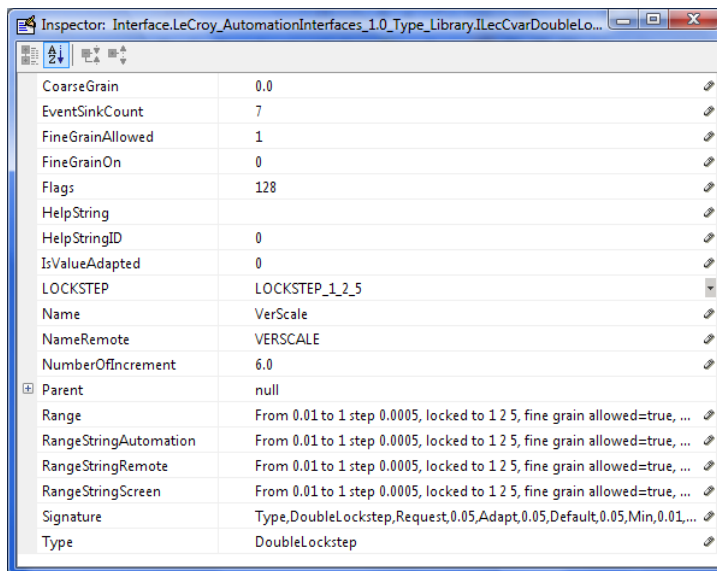
C) Creation of object variable to a waveform **Result** object

```
>> C1_Result = C1.Out.Result
C1_Result =
    Interface.LeCroy_ResultAutomationInterfaces_1.0_Type_Library.ILecAutomationWformResult
```

Note the slight difference in the interface between B and C; B is a ParamResult, while C is a WformResult

D) The **inspect** function can be used to quickly see the property values of a CVAR:

```
>> inspect(c1.Item('VerScale'))
```



Conclusion

Every programming language has syntax requirements, and learning the correct syntax can often be challenging and time consuming. The time required to learn LeCroy X-Stream COM object programming with MATLAB can be reduced by taking time to understand the underlying methodology of both the LeCroy X-Stream object hierarchy and MATLAB software environment. For more information, see both MATLAB and X-Stream documentation.