



Instrument Control from Visual Basic

[Back to Document](#)

Are you looking for an easy way to implement instrument interchangeability in your Visual Basic programs or at least minimize your rework when you replace or upgrade an instrument in your system? This has long been a goal of test and measurement engineers around the world, and they have created their own coalition in search of solutions. Their solution is known as IVI (Interchangeable Virtual Instruments) and the coalition is known as the IVI Foundation (www.ivifoundation.org). This application note focuses on the following instrumentation topics:

Instrument control – An overview about traditional instrument drivers and the evolution to IVI class drivers and IVI instrument-specific drivers.

Measurement Studio IVI ActiveX controls – A discussion of how you can take advantage of the IVI ActiveX controls, which offer a novel and intuitive approach for controlling instruments like oscilloscopes and digital multimeters (DMMs) in Visual Basic.

Controlling instruments with instrument-specific drivers – If you cannot take advantage of the IVI architecture, learn how to control instruments from Visual Basic with instrument-specific drivers.

Note: Measurement Studio includes tools to build measurement applications in ANSI C, Visual C++, and Visual Basic. The Measurement Studio Visual Basic tools were formerly known as ComponentWorks. The IVI ActiveX controls are available in the Measurement Studio Full Development System.

Table of Contents:

- [Instrument Control](#)
- [Easy Instrument Control in Visual Basic – IVI ActiveX Controls](#)
- [Controlling Instruments with Instrument-Specific Drivers](#)
- [Compiling, Modifying, or Creating Instrument Drivers](#)
- [Conclusion](#)

Instrument Control

In the early days of computer-based test and measurement systems, you had to send message-based commands directly from your instrument programs to communicate with message-based devices. For example, to get a value from a digital multimeter might require six separate, often cryptic commands. As computer-based instrumentation became more popular, many developers realized that they could achieve more flexibility through high-level, generic, modular routines that they could reuse in different programs for a particular instrument. Rather than calling those six commands each time you want to read from the multimeter, you can just call a single read function. These libraries of routines – called instrument drivers – enabled you to focus on the application rather than lower-level details.

What Is an Instrument Driver?

An instrument driver is a piece of software that provides an instrument-centric application programming interface (API), while abstracting instrument communication details and enabling you to focus on your primary task – controlling the instrument. Most instrument drivers are written to a specification called *VXIplug&play* (www.vxipnp.org). Although the *VXIplug&play* specification describes the basic format for instrument drivers, it does not define function calls or exact operating behavior. Each instrument has its own instrument driver, so you have to change your programs when you upgrade to a new device or switch device vendors.

Figure 1 shows how you have to make instrument-specific driver calls from your Visual Basic program when you use traditional instrument drivers. In this example, the application reads from a Fluke 45 DMM using the `fl45_Read` instrument driver function. If you substitute a different DMM, then you must modify the application to use the correct function for that instrument.



Figure 1. Although traditional instrument drivers manage low-level details for you, they restrict instrument interchangeability.

What Is IVI?

The IVI Foundation was chartered to achieve instrument interchangeability without requiring application modifications, and their solution is the Interchangeable Virtual Instrument (IVI) specification, an extension to *VXIplug&play*. The IVI specification defines generic instrument classes. The IVI Foundation currently defines the following classes:

- Oscilloscope
- DMM (digital multimeter)
- Arbitrary Waveform/Function Generator
- Switch
- Power Supply

Each class defines functions and attributes common for controlling instruments of that class. For example, the oscilloscope class contains a collection of attributes that are common to all oscilloscopes, such as vertical range, offset, and trigger type. The class also contains functions to set these attributes or retrieve data from the instrument, such as *ConfigureChannel*, *ConfigureTrigger*, and *ReadWaveform*. The IVI class drivers provide a standard programming interface for all instrument drivers that conform to the class specification.

Note: By defining a standard for each oscilloscope function and attribute, the IVI class specifications make it possible for you to write test programs that work with *any* oscilloscope.

Figure 2 shows how, with the IVI architecture, you make calls from your Visual Basic program to the class drivers, which in turn communicate through IVI instrument-specific drivers to the instruments. The IVI instrument-specific drivers contain the information for controlling a particular instrument (for example, a Fluke 45 DMM or a National Instruments NI 5102 oscilloscope), including the command strings, parsing code, and valid ranges of each setting for that particular instrument.

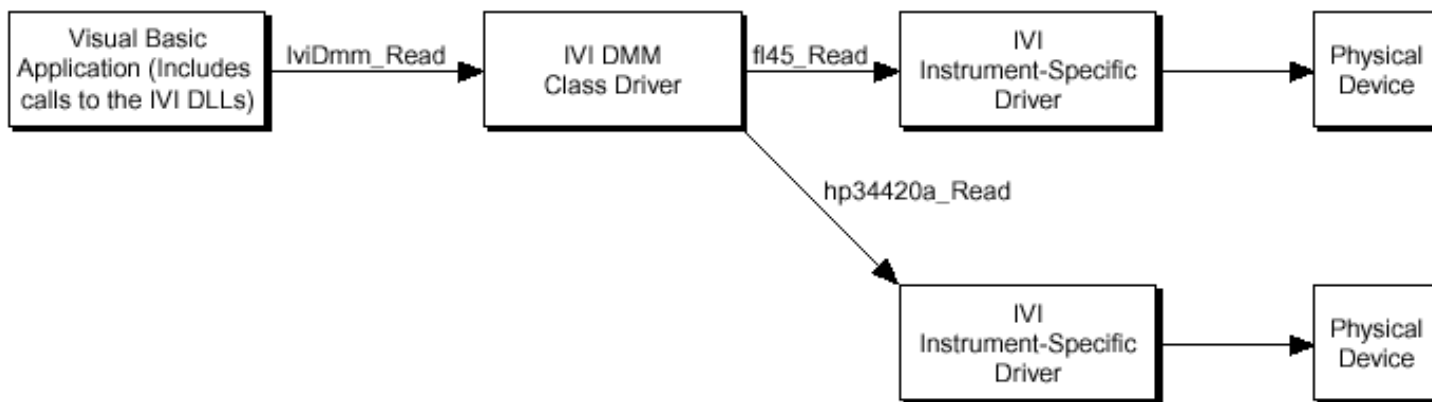


Figure 2. IVI class drivers provide instrument interchangeability.

IVI instrument-specific drivers work the same way as traditional instrument drivers, except that IVI drivers use an attribute-based approach to instrument control, delivering better run-time performance and additional instrument driver capabilities, such as state caching, configurable range checking, configurable status query, simple simulation, and multithread safety.

All instruments are not created equal. For those that offer more features, the IVI Foundation created extension groups. Extensions are functions and attributes that represent more specialized features of an instrument class. For example, although all oscilloscopes have similar fundamental capabilities for vertical and horizontal settings, there is a wide variety of trigger modes across oscilloscopes – video triggers, runt triggers, width triggers, and so on. Through extensions, the IVI Foundation has created standard programming interfaces for features and capabilities that are not standard in every oscilloscope. Therefore, every oscilloscope that accepts video signals will then comply with the video signal extension functions and attributes of the IVI specification.

Easy Instrument Control in Visual Basic – IVI ActiveX Controls

The Measurement Studio IVI ActiveX controls offer design-time configuration through property pages, built-in instrument-like user interfaces, and a Visual Basic friendly API. Because the IVI ActiveX controls actually sit on top of the IVI class drivers, you take advantage of all IVI

benefits – such as interchangeability, simulation, and state caching – without learning the extensive IVI API or using nonnative data types. Figure 3 shows that the IVI control makes all the calls to the IVI class driver for you.

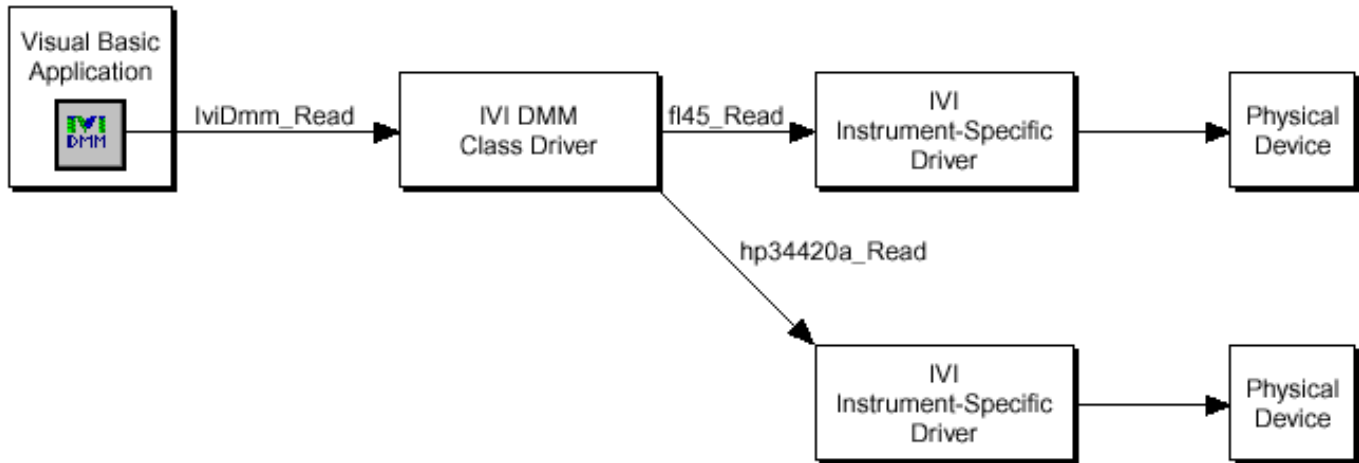


Figure 3. Measurement Studio IVI controls provide quick and easy access to IVI class drivers.

Note: The IVI ActiveX controls are compatible with version 2.0 of the IVI class specifications. Instrument drivers and class drivers written to the 1.0 specification do not work with the controls.

Getting Started – Without Writing Code

The IVI controls offer time-saving features to help you develop your instrument control applications faster and easier in Visual Basic.

Rapidly develop user interfaces with built-in interface styles. Each IVI control features built-in user interface styles that you can change in the property pages or with the **UI Style** property. Although you can make the control invisible at runtime and create your own user interface for the program, you can take advantage of built-in user interfaces to effortlessly create instrument-like front panels. If you want to provide information about what is going on with an instrument without anyone tampering with instrument parameters or settings, use the **Display Only** interface style. For example, you might create a test program with which an operator can monitor but not alter the instrument state, as shown in Figure 4.

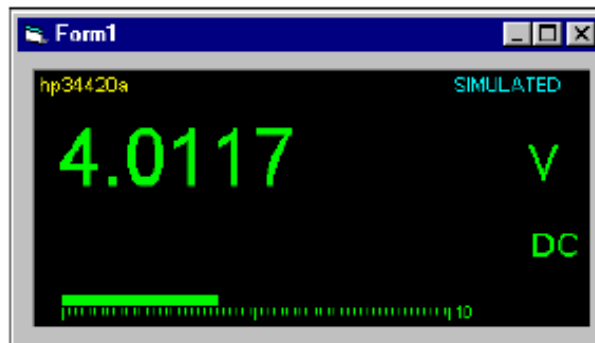


Figure 4. Display Only Interface Style for a DMM Control

If you want to provide an interactive interface for simple instrument control, which is particularly useful as you develop and test your program, use the **Allow Control** interface style. This style combines the **Display Only** style with a set of buttons and knobs for convenient instrument control from the user interface, as shown in Figure 5.

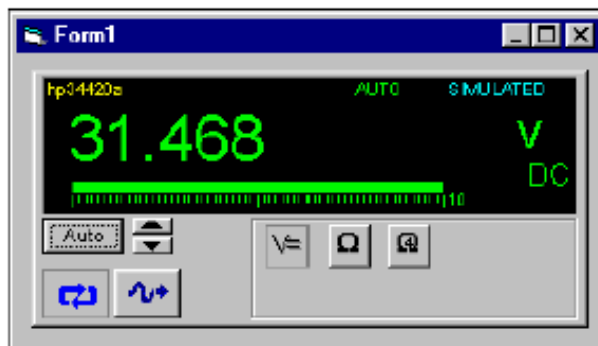


Figure 5. Allow Control Interface Style for a DMM Control

Interactively create reusable configurations during design time. You can create start-up configurations through the property pages during design-time configuration without writing any code. If your measurement tasks require you to change instrument settings, you can create a set of configurations as you design your IVI control and then store them for later use. You also can export and import configurations from a file. For example, if you have an existing configuration for a specific measurement task and you have another program that requires similar functionality, you can import that configuration into your other program.

Tip: You can create, store, load, import, and export configurations programmatically with the `CWIVIConfiguration` object.

Create code-free instrument control applications using Auto Configure. By default, the IVI controls do not start controlling your instruments until you call the `Configure` method. If you want your program to begin instrument control as soon as you run it, select the **Auto Configure** feature in the property pages. When **Auto Configure** is enabled, the IVI control calls `Configure` for you as soon as you run the program, and the program starts with your instruments under computer control. Combine this feature with a built-in user interface style to create a code-free instrument control program, as explained the following example.

Use automatic read intervals to simulate continuous measurements. In most test programs, you probably take a single reading or set of readings from an instrument, but in interactive applications you probably want to continually take measurements. The IVI controls provide the **Autoread Interval** property as a way to emulate a continual measurement behavior. By setting the **Autoread Interval** property to a positive integer n , you are telling the IVI control to perform a read on all configured channels every n milliseconds. Combine this feature with a built-in user interface style and **Auto Configure** to create a simple soft front panel without writing any code, as shown in the following example.

Example –Creating a Soft Front Panel without Writing Code

With soft front panels you can interactively operate your instruments to ensure that they operate correctly and make simple, interactive measurements. For example, an oscilloscope soft front panel is an interactive interface to control any oscilloscope configured to work with the IVI class driver. The soft front panel is a helpful troubleshooting tool when you debug system issues and need to take interactive measurements with your equipment. Because the soft front panel is generic for any oscilloscope that plugs into the class driver, you need to learn how to use it only once for all scopes in your system.

1. Load the `CWIVIScope` control into your Visual Basic project. Place a Scope control on the form, right click on it, and select **Properties**.
2. Select `Scope_CW` as the device logical name. `Scope_CW` is a simulated HP 546xx oscilloscope. If you want to learn more about this or other IVI logical names, click on the **Configure IVI with Measurement & Automation Explorer** icon on the property page.
3. Enable the **Auto Configure** feature. The control automatically calls the `Configure` method with default configuration properties when you run the program.
4. Specify an **Autoread interval** of 1000 ms. The control will invoke the `Read` method for you every second.
5. Click the **Add** button on the **Channels** property page to add a channel. Use the channel's default values.
6. Click **OK**.
7. Run and test your program.

Configuring Instruments

After you place an IVI control on your form, how do you start collecting measurements? From the property pages, select the logical name of

the device. A logical name identifies the particular virtual instrument to use. The virtual instrument, in turn, identifies a particular physical device and specific driver and specifies the initial settings for the session.

Tip: Use National Instruments Measurement &Automation Explorer to create new logical names. You can launch Measurement &Automation Explorer from the property pages – just click on the icon below the list of logical names.

To change properties of a logical name, such as the virtual instrument to which it refers or its inherent attributes, right click on the logical name in Measurement &Automation Explorer, select **Properties**, and then click the **Properties** button.

Next, set configuration properties using the property pages.

Note: You are not physically configuring the instrument when you set properties. The control locally stores all property values and waits for you to call the Configure method. You must call Configure either explicitly in your code or, if you specify configuration settings in the property pages, by selecting the **Auto Configure** property, to actually configure the device.

Call the Configure method to initialize the hardware to match your specified property values. Unlike traditional instrument drivers that offer different configure functions to configure different attributes of the instrument, the IVI controls offer only one Configure method. Having only one Configure method has some benefits:

Simplifies use – You call only one configure method rather than more than one.

Increases performance – The performance of the Configure method relies on the ability of the IVI engine to cache instrument state. State caching ensures that communication to the instrument – often the bottleneck in an instrument control application – is minimized to only those attributes that need to be changed. Remember to enable state caching for your virtual instrument in Measurement & Automation Explorer.

Ensures instrument interchangeability – You call only one Configure method on the control, and then the control calls all of the lower-level IVI class driver configure functions in the appropriate order for you.

Example – Configuring a DMM

Suppose that you want to configure a DMM with its default property page settings when you click a **Configure** button on the user interface. However, you also want to change the measurement function of the DMM to measure 2-wire resistance rather than DC volts (the default measurement function) during runtime by flipping a switch on the user interface. Figure 6 shows this configuration example.

Note: When you change a configuration setting during runtime, you have to call Configure before you can retrieve measurements with that configuration setting.

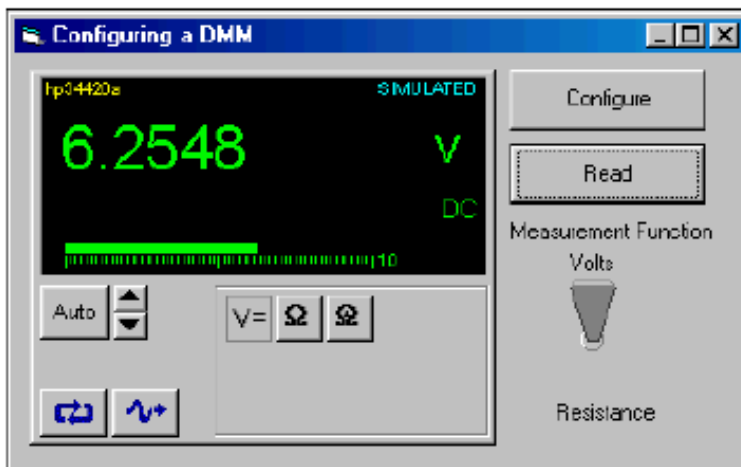


Figure 6. Configuring a DMM

1. Place a CWIviDmm control and a command button named cmdConfigure on the form.
2. Add the following event procedure to your program:

```
Private Sub cmdConfigure_Click()
    CWIviDmm1.Configure
End Sub
```

3. Add a command button named cmdRead and the following event procedure to retrieve the measurement:

```
Private Sub cmdRead_Click()
    CWIviDmm1.Read
End Sub
```

4. Run your program. Click **Configure** and then **Read**. You can click **Read** more than once to retrieve multiple measurements.

5. To add the ability to switch measurement functions from the user interface during runtime, add a Measurement Studio Button (CWButton) control and the following event procedure:

```
Private Sub CWButton1_ValueChanged(ByVal Value As Boolean)
    If (Value = True) Then
        CWIviDmm1.MeasFunction = cwividmmDCVolts
    Else
        CWIviDmm1.MeasFunction = cwividmm2WireResistance
    End If
End Sub
```

6. Run your program. Click **Configure** and then **Read**. You are still collecting measurements in volts. Now switch to measure resistance and click **Read** again. Notice that your measurements are still in volts because you didn't reconfigure the instrument. Click **Configure** and then **Read**. Now your measurements are returned as ohms.

Note: If the Configure method fails because of an invalid property value, you might have tried to set an unsupported extension or property value. You can verify valid property values in the Measurement Studio Reference. To determine if a value is valid for a specific driver, find the corresponding attribute name in the Measurement Studio Reference (if a property has a corresponding attribute, the attribute is listed in the Remarks section of the topic) and search the instrument driver documentation for that attribute name.

Retrieving Data

After you configure the instrument, you can retrieve data. The IVI class specifications define several functions to retrieve different types of data or measurements. The IVI controls simplify data retrieval because one method can return different data types and measurements, depending on what you request. For example, you might request an oscilloscope to return a single measurement like frequency, a one-dimensional array of points representing a waveform, or a 2-dimensional array of points representing a min-max waveform. The following line of code returns the frequency of the data on channel 1:

```
CWIviScope1.Channels(1).Read cwiviscopeMeas_Frequency
```

Notice how you can call the same Read method but return a min-max waveform (two-dimensional array):

```
CWIviScope1.Channels(1).Read cwiviscopeWaveform_MinMax
```

Read versus Fetch

The IVI class specifications define a read as two distinct operations – start a measurement (Initiate) and retrieve the data from the measurement (Fetch). When you call the Read method, the IVI control initiates and retrieves a measurement from the instrument. Each subsequent call to Read initiates and retrieves a new measurement from the instrument. For example, to initiate a DMM, retrieve a measurement, and display it, use the following code:

```
CWIviDmm1.Read
```

To initiate a scope and retrieve the measurement from the first channel, use the following code:

```
CWIviScope1.Channels(1).Read cwiviscopeWaveform
```

The previous line of code actually initiates and samples data from all configured channels but returns only the requested channel (in this example, the first channel). If you want to initiate and return a measurement from both channels on a 2-channel scope, call Read on the first channel to initiate all channels and return the measurement on the first channel and then use the Fetch method to return data on the second channel, as shown in the following code:

```
CWIViScope1.Channels(1).Read cwiviscopeWaveform
CWIViScope1.Channels(2).Fetch cwiviscopeWaveform
```

You also might call Fetch to retrieve different information from the same acquisition. In the following example, Waveform1 and Waveform2 both contain a 1-dimensional array of points representing the acquired waveforms on those channels. Freq1 contains a double precision number representing the frequency of the waveform on Channel 1. Voltage contains the average in volts measured over the entire waveform.

Note: The scope must support the waveform measurement extension to find the frequency and voltage.

```
Waveform1 = CWIViScope1.Channels("Channel1").Read(cwiviscopeWaveform)
Waveform2 = CWIViScope1.Channels("Channel2").Fetch(cwiviscopeWaveform)
Freq1 = CWIViScope1.Channels("Channel1").Fetch(cwiviscopeMeas_Frequency)
Voltage = CWIViScope1.Channels("Channel2").Fetch(cwiviscopeMeas_VoltageAverage)
```

Example – Retrieving Data from a 2-Channel Scope

Suppose you use a 2-channel oscilloscope to take measurements. Your program requires that you obtain waveforms from both channels as well as read the frequency information from both channels all from a single trigger condition – clicking the command button. In this section, you will build the example shown in Figure 7.

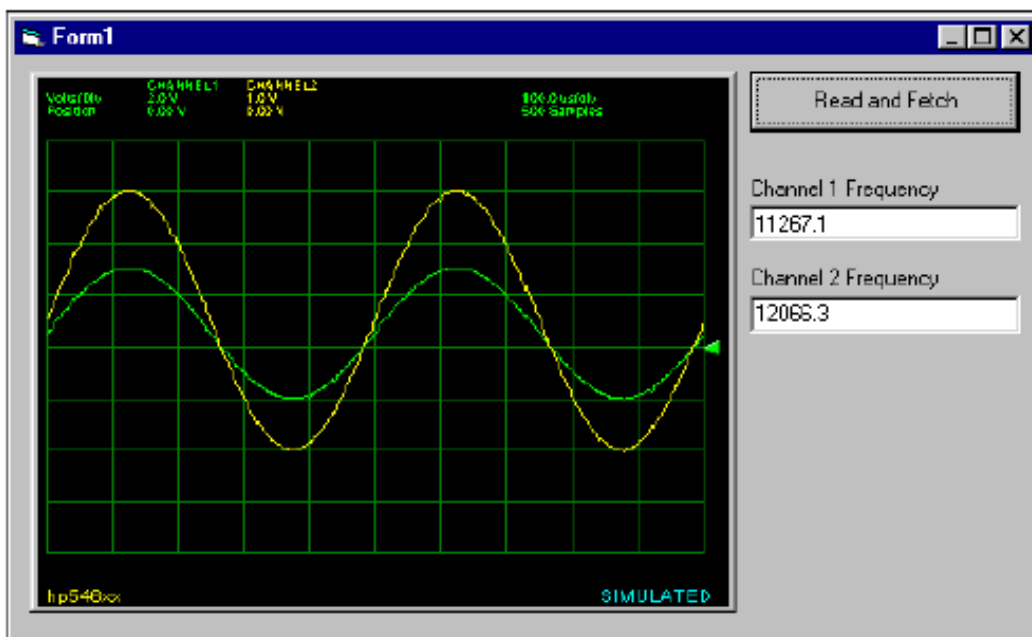


Figure 7. Interface for Retrieving Data from a 2-Channel Oscilloscope

1. Load the CWIViScope control into your Visual Basic project. Place a control on the form, right click on it, and select **Properties**.
2. On the CW IVI page, select Scope_CW as the logical name. Scope_CW is a simulated IVI scope. You can inspect its properties in Measurement & Automation Explorer. Change the **UI Style** to **Display Only**.
3. On the **Channels** property page, add two channels. On the first channel, change the **Vertical Range** to 16.
4. Create the user interface as shown in Figure7. Name the command button **cmdRead** and the two text boxes **txtChanFreq1** and **txtChanFreq2**.
5. Add code so that the instrument is configured when you run the program, as shown in the following event procedure:

```
Private Sub Form_Load()
    CWIViScope1.Configure
End Sub
```

6. Add code to read and display the measurement from each channel and then fetch the frequency from each channel:

Private Sub cmdRead_Click()

```
CWIVI_Scope1.Channels(1).Read cwiviscopeWaveform
CWIVI_Scope1.Channels(2).Fetch cwiviscopeWaveform
txtChanFreq1.Text = _ CWIVI_Scope1.Channels(1).Fetch(cwiviscopeMeas_Frequency)
txtChanFreq1.Text = _ CWIVI_Scope1.Channels(2).Fetch(cwiviscopeMeas_Frequency)
End Sub
```

The call to Read initiates the measurement, which means that the scope samples data *for all configured channels*. After the measurement is complete, the scope sends the data only for the requested channel. In this example, you use the Read method to return the waveform from channel 1 and the Fetch method to return the waveform from channel 2. To retrieve additional information – in this example, the frequency on each channel – call Fetch to return information from the instrument without causing it to get another sample set.

7. Run and test your program.

Handling Instrument Sessions

The IVI engine communicates with instruments from your program via *instrument sessions*. A session, which you can think of as a handle from your program through the IVI engine to the instrument, is a collection of data structures maintained by the IVI engine necessary for your program to communicate with the instrument. Without the Measurement Studio IVI controls, you would call an initialize (init) function to open the session and a close function to release the session and any memory or handles allocated to your instrument control operation.

The Measurement Studio IVI controls manage instrument sessions for you. After you call any method that requires an initialized session handle (such as Read and Fetch), the IVI control automatically initiates the session with default parameters. When you stop your program, the IVI control automatically closes the session for you.

Tip: If you want to change the default behavior of the automatic session initialization, you can explicitly call Init with the appropriate parameters before calling any other methods. For example, the following code initializes the instrument, performs an identification query, and, if there is an active session to the current logical name, closes that session and opens a new one:

```
CWIVI_Scope1.Init True, , , cwiviCloseAndInit
```

If you want control over the session lifetime, explicitly call the CloseSession method. Refer to the Measurement Studio Reference for complete information on Init, CloseSession, or any other property, method, or event.

Calling Utility Functions

The suite of Measurement Studio IVI controls features a utility control called CWIVI_Tools. This control provides access to some low-level IVI functionality that is not included in the CWIVI_Scope, CWIVI_Fgen, CWIVI_DCPower, CWIVI_Switch, CWIVI_SwitchScan, and CWIVI_Dmm controls and includes some application helper functions.

You can use the functions in CWIVI_Tools with any Measurement Studio IVI class control. For some functions, you need to pass a session handle to specify which instrument session is making the call. The IVI controls provide a SessionHandle property that you can pass to utility functions on the CWIVI_Tools control. For example, to query a DMM for its current range setting you could call the GetAttributeValue method with the SessionHandle property:

```
Range = CWIVI_Tools1.GetAttributeValue(CWIVI_Dmm1.SessionHandle, IVIDMM_ATTR_RANGE)
```

If you want to call a function that is not included in the class specification from a specific driver, you can pass the session handle used by the IVI control to any of the instrument driver functions and take advantage of both the specific driver and IVI control features.

Accessing IVI Extensions

The IVI controls support the IVI class extensions. Refer to the Extensions property in the Measurement Studio Reference for information about enabling extension groups. Remember that when you enable an extension, you add additional capability and configuration overhead. Enable only those extensions that you need for your application.

Controlling Instruments with Instrument-Specific Drivers

For some instrument control programs, you might not want to use the IVI-defined instrument classes – perhaps your instrument driver is not

IVI compliant, you have modified an existing driver, or designed your own instrument driver for your project. To control a specific instrument from Visual Basic without using IVI, add a reference to the instrument driver DLL from your Visual Basic project and programmatically control the instrument using the API defined in the instrument driver DLL:

1. Select **Project»References**.
2. Press the **Browse** button in the References dialog and then navigate to the location of the instrument driver DLL. Most National Instruments instrument driver DLLs are installed by default in c:\VXI\inp.
3. Select the instrument driver DLL to add to your project.
4. After you add a reference to your project for an instrument driver DLL, use the Object Browser to view the functions and parameters available in each instrument driver. The functions in each instrument driver should be listed and described in the corresponding help file installed with each driver. Now use any of the functions in the instrument driver.

Example

Suppose you have an instrument-specific driver for a Fluke45 digital multimeter, and you want to create a very simple Visual Basic program that takes a measurement from the DMM and displays the value on the user interface. The instrument driver DLL defines the following functions:

Initialize – The FL45_InitWithOptions function opens a new instrument session to the specified instrument and you can set session attributes such as range checking, state caching, and simulation. You also can initialize the instrument state and query the instrument status with this function.

Measure – The FL45_Measure function initiates the measurement, waits until the DMM has returned to the idle state, and returns the measured value.

Close – The FL45_Close closes the instrument I/O session, destroys the instrument driver session and all of its attributes, and deallocates any memory resources the driver uses.

Tip: To find information about the functions available in an instrument driver DLL, check the help file that came with the DLL. Usually, the first topic contains a hierarchal list of the driver. From there, you'll have to use a little intuition. Although IVI-compliant instrument drivers are written to a very detailed specification, not all traditional instrument drivers are, which means you will find different function names used in different drivers even for instruments of the same class.

1. Create a reference to the instrument driver DLL from your Visual Basic project.
2. Create a simple user interface on the Visual Basic form. Use a Measurement Studio NumEdit (CWNumEdit) control (or a Visual Basic text box) and a command button, as shown in Figure 8.

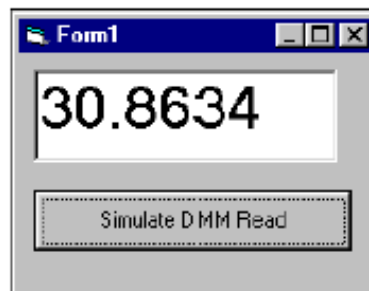


Figure 8. User Interface to Monitor DMM Values

3. Add the following code:

Option Explicit

Dim SessionHandle As Long

Dim ReadVal As Double

Private Sub Form_Load()

 'Initialize the Fluke with no inherent IVI attributes set

 FL45_InitWithOptions "GPIB0::13::InStr", 0, 0, "", SessionHandle

End Sub

```
Private Sub Command1_Click()
    'Get the measurement and display it in the num edit control
    FL45_Measure SessionHandle, FL45_VAL_FREQ, 5000, ReadVal
    CWNumEdit1.Value = ReadVal End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Close instrument session
    FL45_close SessionHandle
End Sub
```

4. Run and test your program.

Tip: If you build this example with an IVI-compliant instrument driver, you can have your IVI instrument driver simulate data so you can test your program. You can implement simulation in Measurement &Automation Explorer under the virtual instrument inherent properties or with the `InitWithOptions` function, as in the following code:

```
FL45_InitWithOptions "GPIB0::13::InStr", 0, 0, "Simulate=1", SessionHandle
```

Compiling, Modifying, or Creating Instrument Drivers

If you choose to use your own instrument drivers to control instruments from Visual Basic, National Instruments offers tools to help you compile or modify existing driver files and create new instrument drivers.

Compiling Instrument Drivers into DLLs

To use an instrument driver from Visual Basic, you need a DLL. You might receive some drivers precompiled, but sometimes you have to compile your own DLLs. For example, if you download instrument drivers from the National Instruments Web site, you get three LabWindows/CVI files – a function panel file (.fp), a source code file (.c), and a header file (.h). The names of these files must be the same except for the different extensions. Use the Instrument Driver Factory (IDF) – a wizard that automates the National Instruments LabWindows/CVI development environment – to automatically compile instrument driver DLLs from LabWindows/CVI files.

Note: You must have Measurement Studio for Visual Basic and LabWindows/CVI – both components of Measurement Studio – installed to use the IDF.

1. Select **Add-Ins»Instrument Driver Factory** in Visual Basic to launch the IDF. If the IDF is not listed, select **Add-Ins»Add-In Manager** to add the IDF to the **Add-Ins** menu. If you're not working in Visual Basic, start the IDF from the Windows Start menu.
2. Select **Use LabWindows/CVI to compile instrument drivers into DLLs with associated WinHelp files** and click **Next**.
3. Select the instrument drivers you want to compile from the list of available drivers. By default, the IDF searches for instrument driver source code in the `\CVI\Instr` directory and the `VXI\plug&play` directory (typically located in `\vxi\pn`). If your instrument driver is located in another location, you can click **Search Paths** to add the path for your instrument driver. Click **Next** to continue.
4. Select the location where you want the DLL and help files copied after the compilation is complete. The IDF automatically deletes all intermediate files from your system. Click **Next** to continue.
5. Verify the information and click **Finish** to launch LabWindows/CVI and automatically compile the instrument driver(s).

When the IDF finishes compiling the instrument drivers, it lists the DLLs and help files that it created. If the IDF was not able to create a DLL or help file, it displays an error.

Modifying Instrument Drivers

If you want to modify an existing instrument driver, use the IDF to launch LabWindows/CVI and load the instrument driver project. After you make modifications, compile the instrument driver files into a DLL. Refer to your LabWindows/CVI documentation for information about the environment or working on instrument drivers.

Creating Your Own Instrument Drivers

Sometimes you might want to create your own instrument drivers from scratch. For example, you might have an instrument that you want to control in a unique way. You can create your own IVI-compliant instrument driver in LabWindows/CVI using the *LabWindows/CVI Instrument Driver Developers Guide*. This guide was written for users who develop instrument drivers to control programmable GPIB, VXI, and RS232 instruments.

See Also:

[Instrument Driver Developers Guide](#)

Conclusion

The IVI Foundation created the IVI architecture to increase instrument interchangeability and reduce rework for test and measurement engineers. With design-time configuration through property pages, built-in instrument-like user interfaces, and a Visual-Basic-friendly API, National Instruments Measurement Studio offers an easy way to integrate IVI architecture into your Visual Basic programs. For more information about IVI, visit the IVI Foundation at www.ivifoundation.org. For more information about Measurement Studio, visit ni.com/mstudio.

[My Profile](#) | [Privacy](#) | [Legal](#) | [Contact NI](#) © 2005 National Instruments Corporation. All rights reserved.