Using IVI Drivers in LabVIEW

Kamran Shah

Introduction

Instrument drivers—software modules that control programmable instruments—have advanced technologically in many ways. Standardization of hardware interfaces, including the standardization of GPIB through the IEEE 488.1 and 488.2 standards, has led to continual improvement of instrument driver technologies. The VXIplug&play Systems Alliance introduced VXIplug&play instrument drivers, which standardized naming conventions, file formats, and distribution of drivers. IVI drivers (Interchangeable Virtual Instruments) are the result of the efforts of the IVI Foundation, a consortium of instrument vendors, systems integrators and end users who have a common interest in further improving the quality and performance of instrument drivers.

This application note targets LabVIEW test developers who are familiar with traditional VXIplug&play instrument drivers but want to migrate to the new IVI drivers. This application note briefly covers the new IVI features, explains how to access the features in LabVIEW, and lists the visible differences between IVI and legacy drivers. You should know basic GPIB, serial and/or VXI concepts and you should know how to operate LabVIEW to develop test applications with traditional drivers based on the VISA VIs. This application note focuses on instrument-specific drivers, not on IVI class drivers. For additional information on IVI class drivers, which enable you to develop hardware-independent applications, please refer to Application Note 121, Using IVI Drivers to Build Hardware-Independent Test Systems with LabVIEW and LabWindows/CVI. You can access this document through the following Web site: http://www.natinst.com/appnotes.nsf/.

IVI Benefits and Features

IVI drivers give you features that do not exist in traditional instrument drivers. This section describes the following IVI benefits and features:

- Standardized Configuration utility
- Standardization benefit
- State caching feature
- Range checking feature
- Simulation feature
- Status checking feature
- Coercion recording feature

The Using IVI Drivers section describes two methods you can use to access the IVI driver features.
Standardized Configuration Utility

You can configure IVI drivers settings independently of a test application. With the IVI component of the Measurement and Automation Explorer (MAX) you can set up virtual instruments for instrument driver settings that an application can use. Instead of using a standard VISA resource name string, such as `GPIB::2::INSTR`, you can pass a previously configured virtual instrument name, such as `SCOPE1`. The IVI engine uses the predefined settings associated with the resource name string `SCOPE1` for the test application. This virtual approach allows you to set up a number of different instrument driver configurations, that test applications can easily access, using the resource name string.

Standardization

You can think of the IVI specifications as supersets of the VXI plug&play standards for driver development. IVI addresses areas that the VXI plug&play standard does not handle. The VXI plug&play Systems Alliance standardizes design requirements with the VXI plug&play driver standard, including some naming convention, data format, and driver distribution standards. These traditional VXI plug&play drivers emphasize ease of use and they do not set guidelines for the internal structure of drivers or for the interface provided for similar instruments. The IVI class specification sets guidelines for both the internal structure and for the programmatic interface of similar instruments.

IVI drivers introduce the concept of instrument classes. The IVI class specification divides instruments into functional classes, such as oscilloscopes and digital multimeters. The IVI Foundation defines instrument classes. (For more information on the IVI Foundation, visit the IVI Foundation web site at www.ivifoundation.org). The IVI class specification establishes the characteristics of each class of instruments. The IVI Foundation also specifies the programmatic interface for these different classes of instruments. For example, with IVI all function generator drivers have the same programmatic interface for basic instrument functionality.

With IVI, two drivers for function generators share the same VI names (with the exception of the prefix), the same number of inputs/outputs, and the same terminal pattern. Furthermore, control definitions are the same for the two drivers. The IVI engine always defines measurement parameters consistently and the instrument driver adjusts parameters to account for differences between individual instruments. For example, the output amplitude for the two function generators is always defined to be volts peak to peak, not volts peak or volts rms. Although the interface to each driver VI is the same, IVI customizes the underlying operation of each function in the driver for the specific instrument. By defining standards for each of these functions, IVI makes it possible for you to develop test programs that can work with any oscilloscope. When you have a standard interface to instrument drivers you work more quickly because you do not need to learn a new interface for each new instrument.

Standardization does not restrict your ability to access individual or unique capabilities of a specific instrument. Instrument-specific VIs supplement the standard VIs and provide access to functions that are unique to the instrument.

Standardization in IVI also enables instrument interchangeability in your test system. For further information on building test systems that are independent of any particular brand or type of instrument, refer to Application Note 121, *Using IVI Drivers to Build Hardware-Independent Test Systems with LabVIEW and LabWindows/CVI*. You can access this document through the following Web site: [http://www.natinst.com/appnotes.nsf/](http://www.natinst.com/appnotes.nsf/).

State Caching

IVI drivers maintain the physical state of instruments in a software cache to increase the efficiency and speed of instrument drivers. Focusing on the ease of use, traditional instrument drivers did not always provide optimal performance. With those drivers, high-level VIs might combine a number of instrument settings into a single VI. Calls to these high-level VIs often resulted in transmission of redundant commands to the instrument. In contrast, IVI drivers use state caching to eliminate redundant commands.

In IVI drivers, an attribute represents each area that you can configure in an instrument. High-level driver VIs logically group a number of attributes together. Consider the Configure Standard Waveform VI illustrated in Figure 1. The VI configures the parameters for a function generator. The VI sets the waveform shape, amplitude, offset, frequency, and start phase for a waveform. When the VI performs a frequency sweep, only the frequency parameter changes. It is
redundant to send the other four settings each time you run the VI. With state caching, only the frequency parameter is sent to the instrument each time you run the VI.

The key to state management in IVI drivers is the IVI engine, which controls the reading and writing of attributes to and from instruments. Through state caching, the IVI engine stores a copy of the current instrument setting of each attribute, performing I/O with an instrument only when an attribute’s value changes.

For more information on how the IVI engine manages state caching, refer to Application Note 122, *Improving Test Performance through Instrument Driver State Management*. You can access this document through the following Web site: http://www.natinst.com/appnotes.nsf/.

![Configure Standard Waveform VI for the HP 33120A Waveform Generator](image)

**Figure 1.** Configure Standard Waveform VI for the HP 33120A Waveform Generator

**Range Checking**

IVI drivers verify that values you specify for an attribute are valid. Previous LabVIEW drivers indicated the valid ranges for settings indirectly, through the online documentation for each control. IVI drivers provide this information and verify the entries you have made, if you enable Range Checking. Range checking is enabled by default but you can disable it after you debug your application, in order to increase execution speed.

IVI drivers return an error through the “error out” indicator if you enter an incorrect value. Error checking includes consideration of rounding performed by specific instruments. The driver will coerce or round values to the actual value that the instrument will use internally, to reduce any discrepancies. When you specify an attribute value, IVI drivers also identify dependencies for the attribute ranges on other instrument settings. For example, the vertical range (in volts per division) for an oscilloscope channel is dependent on the attenuation of the probe that you are using. When they calculate the valid vertical range, IVI oscilloscope drivers take into account the attenuation of the probe. The IVI engine uses this dynamically calculated range to verify the setting. This approach to range checking eliminates incorrect settings and points out conflicts in an application.
Simulation

Unlike traditional drivers, IVI drivers have a Simulation mode in which you can make calls to the driver without being connected to an instrument. Simulation mode provides the benefits shown in the following table.

<table>
<thead>
<tr>
<th>Characteristic of Simulation Mode</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides an instrument handle that VIs in the driver can use.</td>
<td>To eliminate run-time errors that invalid instrument handles cause.</td>
</tr>
<tr>
<td>Does not alter any other behavior in driver operation. For example,</td>
<td>1. To verify the values you are planning to send to the instrument, while you develop the test application.</td>
</tr>
<tr>
<td>range checking occurs as it does when a physical instrument is present.</td>
<td>2. To test a new instrument feature before you purchase that instrument, by installing its IVI driver and running a test program to ensure that it provides the functionality that you want.</td>
</tr>
<tr>
<td>Lets you simulate the data that you normally acquire, using built-in</td>
<td>To test your application with realistic data.</td>
</tr>
<tr>
<td>algorithms to simulate data generation.</td>
<td></td>
</tr>
</tbody>
</table>

For more information on simulation, refer to Application Note 120, *Using IVI Drivers to Simulate Your Instrumentation Hardware in LabVIEW and LabWindows/CVI*. You can access this document through the following Web site: [http://www.natinst.com/appnotes.nsf/](http://www.natinst.com/appnotes.nsf/).

Status Checking

Traditional LabVIEW drivers provide error-query VIs that a programmer uses to check the status of instruments. This technique burdens the programmer with inserting error-query VIs throughout the application to verify operation at various stages. With IVI instrument drivers, the user can check the status of an instrument after every function that interacts with the instrument without adding extra VI calls. Status checking in IVI drivers is enabled by default so that users can verify their applications during development. After the application has been thoroughly tested and verified, users can disable status checking to improve performance.

The IVI engine checks the status of an instrument only after a function writes an attribute to, or reads a value from, an instrument. If an instrument error occurs at that time, the IVI engine returns an error through the “error out” control. The developer can conditionally call the error query VI to learn more details about the instrument-specific error. An error reported from status checking does not invalidate the cached state of the instrument.

Recording Coercions

Often you choose an instrument setting from a range of values, for example from 1.0 to 1000.0, and the instrument coerces the value to one of several selected values. A digital multimeter (DMM) instrument might accept a value from 1.0 to 1000.0, and the DMM would coerce that value to one of three maximum reading ranges, 10.0, 100.0, and 1000.0. In this DMM example, if you set the range to 50.0, the instrument would coerce the value to 100.0.

To make state caching work properly, the IVI engine must store the coerced value in the cache. Therefore, instead of letting the instrument coerce the value, the IVI driver coerces the value before it sets the instrument.

If you want to track the coercions that the IVI engine performs, you can enable Record Value Coercions. Record Value Coercions maintains a list of all coercions for Integer and Real values passed to the instrument driver VIs. You can call the Get Next Coercion Record VI which accesses the coercions by retrieving and clearing the oldest recorded instance.

**Note** If you enable Record Value Coercions and never use Get Next Coercion Record VI to retrieve and clear those coercion records, the records build up and could eventually overflow your computer memory.
Using IVI Features

In most cases, you develop a test application with the IVI driver VIs in the same way that you develop a test application with traditional LabVIEW drivers. Like traditional LabVIEW driver VIs, IVI driver VIs are grouped together in functional areas that you combine into an application. Unlike traditional LabVIEW driver VIs, IVI driver VIs operate differently internally, because they rely on the IVI engine to coordinate and control the features described in the IVI Features section. For this reason, IVI drivers communicate with instruments and the IVI engine through DLLs (dynamic link libraries).

The internal differences between traditional drivers and IVI drivers should not affect your use of driver VIs to develop an application. After you configure the IVI driver with the IVI features you want—state caching, range checking, simulation, status checking, or recording coercions—you continue developing an application as you do with a traditional LabVIEW instrument driver.

You can configure the IVI features—state caching, range checking, simulation, status checking, or recording of coercions—through one of the following virtual instruments:

- Initialize With Options VI
- Virtual instruments that you set up with the Measurement and Automation Explorer

Initialize with Options.vi

IVI drivers include two distinct initialization VIs. The first performs like the initialization VI found in legacy drivers and the second is the Initialize With Options VI. Figure 2 shows a sample front panel for the FL45 Initialize with Options VI. The only difference between the Initialize with Options VI and the Initialize VI of a traditional driver is the option string. The option string configures the IVI driver features that you choose: state caching, range checking, simulation, status checking, and record value coercions. For instrument drivers that support a family of instruments, you also can use the option string to set the particular model of instrument that you want the driver to emulate.

![Sample Front Panel for the FL45 Initialize with Options VI](image)

**Figure 2.** Sample Front Panel for the FL45 Initialize with Options VI
To enable one of the IVI features, you set its value to 1 in the option string. To disable a feature, you set its value to 0. The IVI engine uses these settings when it runs your test application. The following table lists IVI driver features, their corresponding strings for the option string, and the default value setting for each feature. The IVI engine uses the default value listed below whenever you do not name a feature in the option string.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Option String</th>
<th>Default Value</th>
<th>Default State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Caching</td>
<td>Cache</td>
<td>1</td>
<td>Enabled</td>
</tr>
<tr>
<td>Range Checking</td>
<td>RangeCheck</td>
<td>1</td>
<td>Enabled</td>
</tr>
<tr>
<td>Simulation</td>
<td>Simulate</td>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>Status Checking</td>
<td>QueryInstrStatus</td>
<td>1</td>
<td>Enabled</td>
</tr>
<tr>
<td>Record Value Coercions</td>
<td>RecordCoercions</td>
<td>0</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

The option string in Figure 2, Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1, results in the following configuration:

- Instrument simulation is disabled.
- Range checking is enabled.
- Status checking is enabled.
- State caching is enabled.
- Recording of coercions of values is disabled.

### Using Measurement and Automation Explorer to Configure IVI

This section discusses the National Instruments Measurement and Automation Explorer (MAX) which is integrated into the Windows Explorer so that you can easily manage National Instruments products and services. The IVI component for MAX comes with the IVI Engine version 1.2 or later. You can download the IVI Engine from the National Instruments Web site.

Instead of the Initialize with Options VI, you can use the IVI Configuration utility in the Measurement & Automation Explorer to configure a virtual instrument to store the settings you want to use. This method utilizes the standard Initialize VI that you use with traditional LabVIEW drivers. To make your program use the configured settings, you pass the name of the configured virtual instrument to the resource name control of the Initialize VI. For example, if an oscilloscope is connected to your PC through the GPIB bus with an address of 2, you can use either of the following approaches:

- As with any legacy driver, you can enter GPIB::2::INSTR for the resource name. When you use this approach MAX settings do not take effect.
- Alternatively you can enter a virtual name that you associate with your oscilloscope as the resource name string.

The settings you make through MAX could conflict with settings you make with the Initialize With Options VI. Therefore, you will generate errors if you attempt to use both approaches. Make sure the option string is empty when you use the MAX approach.
Consider the following example. If you previously configured the fl45 virtual instrument for a Fluke 45 Digital Multimeter, you can set and use the Initialize.VI of the specific driver. Simply replace the resource name string with fl45. Figure 3 illustrates this configuration. The IVI engine then uses the configuration parameters associated with the fl45 virtual instrument when it executes your application.

Figure 3. Using a Virtual Instrument Name to Initialize Driver Settings

In the IVI subfolder of MAX, you find the virtual instruments that you can configure. Figure 4 illustrates five virtual instruments that have been set up in MAX. Along with virtual instruments, you also see subfolders for Logical Names, Instrument Drivers, and Devices. Logical Names are primarily used from the class driver level. They point to Virtual Instruments and allow for interchangeability. Virtual Instruments consist of an instrument driver, property settings and—when you are not in Simulation mode—a device. Instrument drivers specify the instrument driver DLL while the Device properties specify the resource name to access a physical device.

Figure 4. View of the IVI Configuration Entries in MAX
Using the MAX Wizard

Before you configure driver settings, you create a virtual instrument. You can use the wizard that comes with MAX to create your virtual instrument. To invoke the wizard, right-click on the Virtual Instruments subdirectory and select Insert. The wizard guides you through the steps that are necessary to set up the virtual instrument, as summarized in the following graphic.

**Figure 5.** Using the MAX Wizard to Set Up a Virtual Instrument
The following items describe the steps you take to create a virtual instrument in the wizard. Figure 5 illustrates the steps:

- In the first dialog box of the wizard you specify the name you want to give to the virtual instrument. You will specify this name in the resource name string when you initialize an instrument.
- In the second dialog box you specify the specific instrument driver that you want to associate with the virtual instrument. You must create an entry for your instrument driver if a driver does not currently exist in the list of existing specific drivers.
- In the third dialog box you specify the physical device that you want to associate with the virtual instrument. If an instrument is not connected to your PC, you can select None – Simulate A Device to simulate the instrument.

When you click on Finish in the final dialog box that appears, the wizard creates the virtual instrument and displays it in the MAX directory tree, along with the other virtual instruments.

You can enable or disable IVI driver features for a virtual instrument that you create in MAX, as follows:

1. Right-click on the virtual instrument you want to modify and select Properties. A property window appears, as shown in Figure 6, in which you can modify the default settings.

2. Click on the Properties buttons to change a particular setting for a specific driver or device, such as the name of the driver DLL or the GPIB address of the device.

3. Click on the Change buttons to change to another driver or device. For example, you might connect your computer to one Fluke 45 DMM at GPIB address 2 and to another Fluke 45 DMM at GPIB address 5. You can configure two virtual devices called FL45_1 and FL45_2 to correspond to each of the physical devices.

4. In the Inherent Attributes tab shown in Figure 7, select IVI driver features that you want. In Figure 7 state caching, range checking, and querying of instrument status are enabled and Simulation is disabled. Recording of coercions, interchange checking, and spying are also disabled.

   Note The IVI driver features are called inherent attributes within the Measurement & Automation Explorer.
Accessing Individual Instrument Settings

Traditional LabVIEW drivers have block diagrams that you can modify. In contrast, the bulk of the code for an IVI driver resides in a dynamic link library (DLL). Therefore, you cannot access the source code directly from the block diagram.

The configuration functions within the DLL combine logical groupings of instrument settings into one function. Although it is possible to access these settings directly through LabVIEW using the property node, it is recommended that you control your instrument through instrument driver VIs. Driver developers address dependencies, sequencing, and performance when they write a driver for a specific instrument. For example, developers can choose the best sequence or order in which attributes should be accessed. For this reason, it is recommended that you use the VIs provided with the driver rather than accessing the individual settings. If you have state caching enabled, controlling the instrument through these VIs adds a trivial amount of performance overhead.

Differences with Traditional Drivers

This section addresses some of the changes that you will notice with IVI drivers if you are familiar with traditional LabVIEW drivers.

VISA Sessions versus Instrument Handles

In most traditional LabVIEW drivers, a VISA session serves as an identifier for communication with GPIB, serial, and VXI instruments; you create a VISA session when you initiate communication with an instrument, and you close the session when you end communication. Your application passes a VISA session between VIs.

In IVI drivers, an instrument handle serves as the identifier. You create this instrument handle in one of your initialize functions. Instead of passing VISA sessions between VIs, your application passes the instrument handle from VI to VI.
Text Menus versus Icons

Palettes for IVI drivers use text menus by default, instead of icons. When you work with drivers, you can often recognize texts names more easily than icons. Menu structures and naming conventions for IVI drivers are similar throughout a class of instruments, so the menu view provides familiarity and consistency to users. If you prefer the traditional icon view, LabVIEW permits you to change to icon view. Figure 8 compares the two different menu views. The HP34401 Multimeter illustrates the icon view, and the Fluke 45 DMM illustrates the text view.

![Figure 8. Menu and Icon Palettes for IVI Drivers](image)

New VIs

A few new VIs exist for IVI drivers that do not exist for traditional instrument drivers.

**Initialize with Options VI**—Discussed previously in this document, this VI is identical to the Initialize VI except for the addition of the option string control. You use the option string to configure the inherent settings of the driver, such as simulation and range checking.

**Get Next Coercion Record VI**—Tracks input values that the IVI driver coerces to new values. Get Next Coercion Record VI returns and clears coercions, if any, when you enable Record Coercions.

![Note](image) If you enable Record Value Coercions and never use Get Next Coercion Record VI to retrieve and clear those coercion records, the records build up and could eventually overflow your computer memory.
**Conclusion**

Instrument drivers are an indispensable tool to help you rapidly develop test and measurement applications. By providing high-level VIs for programming, they eliminate the need for application developers to learn complex programming protocols. The new technologies introduced with IVI drivers maintain the benefits of traditional drivers while adding features that improve the application development process and application performance. You should develop an application with IVI instrument drivers in the same way that you develop applications with traditional LabVIEW drivers, except that you can also take advantage of the new IVI features described in this document.

**Bibliography**


You can access the above-referenced application notes through the following Web site: 