



# ***Getting Started with IVI Drivers***

**Your Guide to Using IVI with  
LabVIEW™**

**Version 1.2**

**© Copyright IVI Foundation, 2015  
All rights reserved**

---

*The IVI Foundation has full copyright privileges of all versions of the IVI Getting Started Guide. For persons wishing to reference portions of the guide in their own written work, standard copyright protection and usage applies. This includes providing a reference to the guide within the written work. Likewise, it needs to be apparent what content was taken from the guide. A recommended method in which to do this is by using a different font in italics to signify the copyrighted material.*



# Contents



<b>Chapter 1</b>	<b>Introduction.....</b>	<b>4</b>
	Purpose.....	4
	Why Use an Instrument Driver? .....	4
	Why IVI?.....	5
	Why Use an IVI Driver? .....	7
	Flavors of IVI Drivers .....	8
	Shared Components .....	8
	Download and Install IVI Drivers .....	8
	Familiarizing Yourself with the Driver .....	9
	Examples .....	9
<b>Chapter 2</b>	<b>Using IVI with LabVIEW™ .....</b>	<b>12</b>
	The Environment .....	12
	Example Requirements .....	12
	Download and Install the Driver .....	12
	Using IVI-C.....	12
	Create a VI and Access the Driver .....	13
	Initialize the Instrument.....	14
	Configure the Instrument .....	15
	Take the Reading .....	15
	Display the Reading .....	16
	Close the Session .....	16
	Add Error Checking .....	16
	Run the Application .....	16
	Setting a Property in an IVI-C Driver .....	17
	Using IVI-COM.....	17
	Create a VI and Access the Driver .....	17

Initialize the Instrument .....	19
Configure the Instrument.....	20
Take the Reading .....	21
Display the Reading .....	21
Close the Driver and Automation Sessions.....	21
Add Error Checking .....	21
Run the Application .....	22
Further Information.....	12



# Chapter 1

## Introduction

•••

### Purpose

Welcome to ***Getting Started with IVI Drivers: Your Guide to Using IVI with LabVIEW™***. This guide introduces key concepts about IVI drivers and shows you how to create a short program to perform a measurement. The guide is part of the IVI Foundation's series of guides, ***Getting Started with IVI Drivers***.

***Getting Started with IVI Drivers*** is intended for individuals who write and run programs to control test-and-measurement instruments. Each guide focuses on a different programming environment. As you develop test programs, you face decisions about how you communicate with the instruments. Some of your choices include Direct I/O, VXI *plug&play* drivers, or IVI drivers. If you are new to using IVI drivers or just want a quick refresher on the basics, ***Getting Started with IVI Drivers*** can help.

***Getting Started with IVI Drivers*** shows that IVI drivers can be straightforward, easy-to-use tools. IVI drivers provide a number of advantages that can save time and money during development, while improving performance as well. Whether you are starting a new program or making improvements to an existing one, you should consider the use of IVI drivers to develop your test programs.

So consider this the “hello, instrument” guide for IVI drivers. If you recall, the “hello world” program, which originally appeared in *Programming in C: A Tutorial*, simply prints out “hello, world.” The “hello, instrument” program performs a simple measurement on a simulated instrument and returns the result. We think you'll find that far more useful.

### Why Use an Instrument Driver?

To understand the benefits of IVI drivers, we need to start by defining instrument drivers in general and describing why they are useful. An instrument driver is a set of software routines that controls a programmable instrument. Each routine corresponds to a programmatic operation, such as configuring, writing to, reading from, and triggering the instrument. Instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the programming protocol for each instrument.

Starting in the 1970s, programmers used device-dependent commands for computer control of instruments. But lack of standardization meant even two digital multimeters from the same manufacturer might not use the same commands. In the early 1990s a group of instrument manufacturers developed Standard

Commands for Programmable Instrumentation (SCPI). This defined set of commands for controlling instruments uses ASCII characters, providing some basic standardization and consistency to the commands used to control instruments. For example, when you want to measure a DC voltage, the standard SCPI command is "MEASURE : VOLTAGE : DC ?".

In 1993, the *VXIplug&play* Systems Alliance created specifications for instrument drivers called *VXIplug&play* drivers. Unlike SCPI, *VXIplug&play* drivers do not specify how to control specific instruments; instead, they specify some common aspects of an instrument driver. By using a driver, you can access the instrument by calling a subroutine in your programming language instead of having to format and send an ASCII string as you do with SCPI. With ASCII, you have to create and send the instrument the syntax "MEASURE : VOLTAGE : DC ?", then read back a string, and build it into a variable. With a driver you can merely call a function called `MeasureDCVoltage()` and pass it a variable to return the measured voltage.

Although you still need to be syntactically correct in your calls to the instrument driver, making calls to a subroutine in your programming language is less error prone. If you have been programming to instruments without a driver, then you are probably all too familiar with hunting around the programming guide to find the right SCPI command and exact syntax. You also have to deal with an I/O library to format and send the strings, and then build the response string into a variable.

## Why IVI?

The *VXIplug&play* drivers do not provide a common programming interface. That means programming a Keithley DMM using *VXIplug&play* still differs from programming a Keysight DMM. For example, the instrument driver interface for one may be `ke2000_read` while another may be `ag34401_get` or something even farther afield. Without consistency across instruments manufactured by different vendors, many programmers still spent a lot of time learning each individual driver. To carry *VXIplug&play* drivers a step (or two) further, in 1998 a group of end users, instrument vendors, software vendors, system suppliers, and system integrators joined together to form a consortium called the Interchangeable Virtual Instruments (IVI) Foundation. If you look at the membership, it's clear that many of the foundation members are competitors. But all agreed on the need to promote specifications for programming test instruments that provide better performance, reduce the cost of program development and maintenance, and simplify interchangeability.

For example, for any IVI driver developed for a DMM, the measurement command is *IviDmmMeasurement.Read*, regardless of the vendor. Once you learn how to program the commands specified by IVI for the instrument class, you can use any vendor's instrument and not need to relearn the commands. Also commands that are common to all drivers, such as *Initialize* and *Close*, are identical regardless of

the type of instrument. This commonality lets you spend less time browsing through the help files in order to program an instrument, leaving more time to get your job done.

That was the motivation behind the development of IVI drivers. The IVI specifications enable drivers with a consistent and high standard of quality, usability, and completeness. The specifications define an open driver architecture, a set of instrument classes, and shared software components. Together these provide consistency and ease of use, as well as the crucial elements needed for the advanced features IVI drivers support: instrument simulation, automatic range checking, state caching, and interchangeability.

The IVI Foundation has created IVI class specifications that define the capabilities for drivers for the following thirteen instrument classes:

<b>Class</b>	<b>IVI Driver</b>
Digital multimeter (DMM)	IviDmm
Oscilloscope	IviScope
Arbitrary waveform/function generator	IviFgen
DC power supply	IviDCPwr
AC power supply	IviACPwr
Switch	IviSwth
Power meter	IviPwrMeter
Spectrum analyzer	IviSpecAn
RF signal generator	IviRFSigGen
Upconverter	IviUpconverter
Downconverter	IviDownconverter
Digitizer	IviDigitizer
Counter/timer	IviCounter

IVI Class Compliant drivers usually also include capability that is not part of the IVI Class. It is common for instruments that are part of a class to have numerous functions that are beyond the scope of the class definition. This may be because the capability is not common to all instruments of the class or because the instrument offers some control that is more refined than what the class defines.



IVI also defines custom drivers. Custom drivers are used for instruments that are not members of a class. For example, there is not a class definition for network analyzers, so a network analyzer driver must be a custom driver. Custom drivers provide the same consistency and benefits described below for an IVI driver, except interchangeability.

IVI drivers conform to and are documented according to the IVI specifications and usually display the standard IVI logo.



## Why Use an IVI Driver?

Why choose IVI drivers over other possibilities? Because IVI drivers can increase performance and flexibility for more intricate test applications. Here are a few of the benefits:

**Consistency** – IVI drivers all follow a common model of how to control the instrument. That saves you time when you need to use a new instrument.

**Ease of use** – IVI drivers feature enhanced ease of use in popular Application Development Environments (ADEs). The APIs provide fast, intuitive access to functions. IVI drivers use technology that naturally integrates in many different software environments.

**Quality** – IVI drivers focus on common commands, desirable options, and rigorous testing to ensure driver quality.

**Simulation** – IVI drivers allow code development and testing even when an instrument is unavailable. That reduces the need for scarce hardware resources and simplifies test of measurement applications. The example programs in this document use this feature.

**Range checking** – IVI drivers ensure the parameters you use are within appropriate ranges for an instrument.

**State caching** – IVI drivers keep track of an instrument's status so that I/O is only performed when necessary, preventing redundant configuration commands from being sent. This can significantly improve test system performance.

**Interchangeability** – IVI drivers enable exchange of instruments with minimal code changes, reducing the time and effort needed to integrate measurement devices into new or existing systems. The IVI class specifications provide syntactic

interchangeability but may not provide behavioral interchangeability. In other words, the program may run on two different instruments but the results may not be the same due to differences in the way the instrument itself functions.

## Flavors of IVI Drivers

To support all popular programming languages and development environments, IVI drivers provide either an IVI-C or an IVI-COM (Component Object Model) API. Driver developers may provide either or both interfaces, as well as wrapper interfaces optimized for specific development environments.

Although the functionality is the same, IVI-C drivers are optimized for use in ANSI C development environments; IVI-COM drivers are optimized for environments that support the Component Object Model (COM). IVI-C drivers extend the *VXIplug&play* driver specification and their usage is similar. IVI-COM drivers provide easy access to instrument functionality through methods and properties.

All IVI drivers communicate to the instrument through an I/O Library. Our examples use the Virtual Instrument Software Architecture (VISA), a widely used standard library for communicating with instruments from a personal computer.

## Shared Components

To make it easier for you to combine drivers and other software from various vendors, the IVI Foundation members have cooperated to provide common software components, called IVI Shared Components. These components provide services to drivers and driver clients that need to be common to all drivers. For instance, the IVI Configuration Server enables administration of system-wide configuration.

**Important! You must install the IVI Shared Components before an IVI driver can be installed.**

The IVI Shared Components can be downloaded from vendors' web sites as well as from the IVI Foundation Web site.

To download and install shared components from the IVI Foundation Web site:

- 1 Go to the IVI Foundation Web site at <http://www.ivifoundation.org>.
- 2 Locate Shared Components.
- 3 Choose the IVI Shared Components msi file for the Microsoft Windows Installer package or the IVI Shared Components exe for the executable installer.

## Download and Install IVI Drivers

After you've installed Shared Components, you're ready to download and install an IVI driver. For most ADEs, the steps to download and install an IVI driver are identical. For the few that require a different process, the relevant **Getting Started with IVI Drivers** guide provides the information you need.

IVI Drivers are available from your hardware or software vendor's web site or by linking to them from the IVI Foundation web site.

To see the list of drivers registered with the IVI Foundation, go to <http://www.ivifoundation.org>.

## Familiarizing Yourself with the Driver

Although the examples in *Getting Started with IVI Drivers* use a DMM driver, you will likely employ a variety of IVI drivers to develop test programs. To jumpstart that task, you'll want to familiarize yourself quickly with drivers you haven't used before. Most ADEs provide a way to explore IVI drivers to learn their functionality. In each IVI guide, where applicable, we add a note explaining how to view the available functions. In addition, browsing an IVI driver's help file often proves an excellent way to learn its functionality.

## Examples

As we noted above, each guide in the *Getting Started with IVI Drivers* series shows you how to use an IVI driver to write and run a program that performs a simple measurement on a simulated instrument and returns the result. The examples demonstrate common steps using IVI drivers. Where practical, every example includes the steps listed below:

- Download and Install the IVI driver— covered in the Download and Install IVI Drivers section above.
- Determine the VISA address string – Examples in *Getting Started with IVI Drivers* use the simulate mode, so we chose the address string **GPIB0::23::INSTR**, often shown as GPIB::23. If you need to determine the VISA address string for your instrument and the ADE does not provide it automatically, use an IO application, such as National Instruments Measurement and Automation Explorer (MAX) or Agilent Connection Expert.
- Reference the driver or load driver files – For the examples in the IVI guides, the driver is the **IVI-COM/IVI-C Version 1.2.5.0 for 34401A, April 2013 (from Keysight Technologies)** . . . or the **Keysight 34401A IVI-C driver, Version 4.5, January 2015 (from National Instruments)**.
- Create an instance of the driver in ADEs that use COM – For the examples in the IVI guides, the driver is the **Agilent34401 (IVI-COM) or hp34401a (IVI-C)**.
- Write the program:
  - Initialize the instrument – Initialize is required when using any IVI driver. Initialize establishes a communication link with the instrument and must be called before the program can do anything with the instrument. We set reset to **true**, ID query to **false**, and simulate to **true**.

Setting reset to true tells the driver to initially reset the instrument. Setting the ID query to false prevents the driver from verifying that the connected instrument is the one the driver was written for. Finally, setting simulate to true tells the driver that it should not attempt to connect to a physical instrument, but use a simulation of the instrument.

- Configure the instrument – We set a range of **1.5 volts** and a resolution of **0.001 volts (1 millivolt)**.
- Access an instrument property – We set the trigger delay to **0.01 seconds**.
- Set the reading timeout – We set the reading timeout to **1000 milliseconds (1 second)**.
- Take a reading
- Close the instrument – This step is required when using any IVI driver, unless the ADE explicitly does not require it. We close the session to free resources.

**Important! Close may be the most commonly missed step when using an IVI driver. Failing to do this could mean that system resources are not freed up and your program may behave unexpectedly on subsequent executions.**

- Check the driver for any errors.
- Display the reading.

**Note:** *Examples that use a console application do not show the display.*

Now that you understand the logic behind IVI drivers, let's see how to get started.



## Chapter 2

# Using IVI with LabVIEW™

• • •

### The Environment

National Instruments LabVIEW is a graphical development environment for signal acquisition, measurement analysis, and data presentation. LabVIEW provides the flexibility of a programming language with less complexity than traditional development tools.

### Example Requirements

- LabVIEW 2011 or later
- IVI-C: Keysight 34401A IVI-C driver, Version 4.5, January 2015 (from National Instruments)
- IVI-COM: Keysight 34401A IVI-COM/IVI-C driver, Version 1.2.5.0, April 2013 (from Keysight Technologies)

**Note:** *These drivers may require an I/O library to be installed. Check the driver vendor's Web site for details.*

**Note:** *IVI-C driver requires the NI IVI Compliance Package to be installed. Check National Instruments Web site for details.*

### Download and Install the Driver

If you have not already installed the driver, go to the vendor Web site and follow the instructions to download and install it.

Since LabVIEW supports both IVI-C and IVI-COM drivers, this example is written two ways, first to show how to use an IVI-C driver in LabVIEW, and second how to use an IVI-COM driver in LabVIEW.

### Using IVI-C

All IVI-C drivers provide a Dynamic Link Library (DLL) interface. While LabVIEW provides the Call Library Function node to call DLLs, many IVI-C drivers also come with a LabVIEW wrapper that provides the familiar VI interface to the driver's functions, making it easier to use in LabVIEW. If your IVI-C driver does not have a LabVIEW wrapper, you can create one using a free tool by clicking on **LabVIEW Instrument Driver Import Wizard** at:

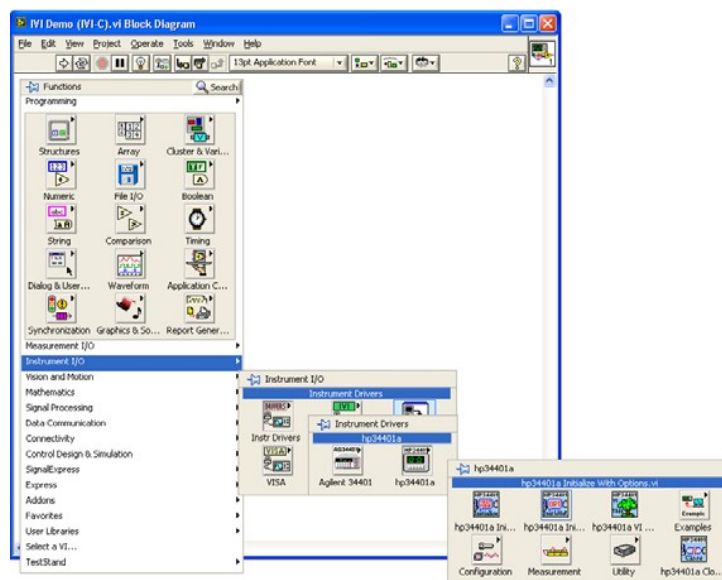
<http://www.ni.com/devzone/idnet/development.htm>.

**Note:** *The functionality shown in this section is available in a LabVIEW example supplied with the IVI driver from National Instruments.*

## Create a VI and Access the Driver

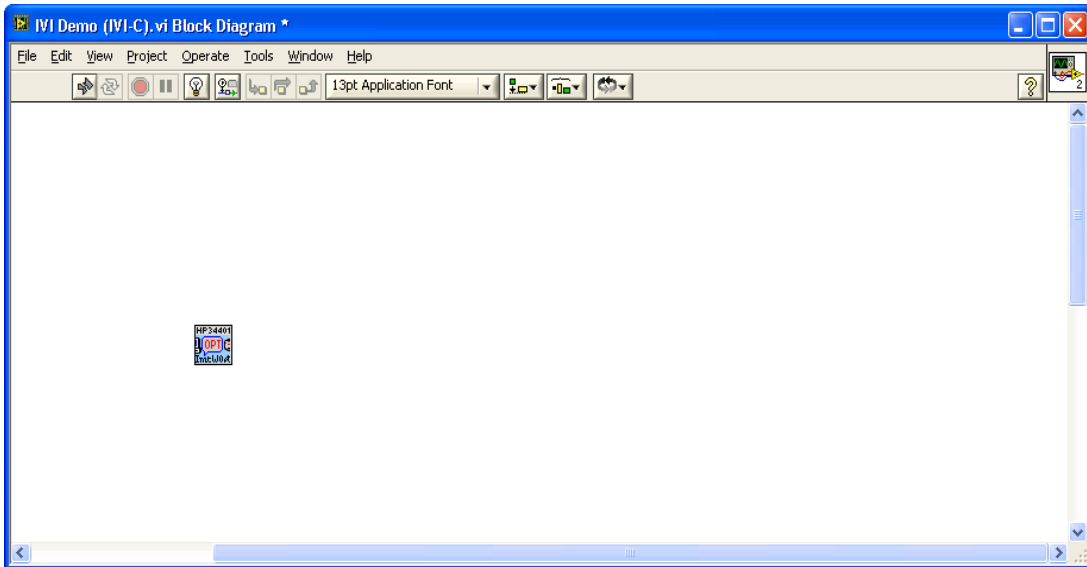
- 1 Launch LabVIEW.
- 2 From the File menu, select New VI. The Front Panel and Block Diagram appear.
- 3 Right-click in the Block Diagram. The Functions palette appears.
- 4 Select the Instrument I/O subpalette and then the Instrument Drivers subpalette. You can access all instrument driver VIs from this palette.
- 5 Click Instrument Drivers. Select hp34401a from the palette.
- 6 Select the hp34401a IVI driver from the palette.

**Note:** If the driver you want to use is not listed, download and install the driver, and close and restart LabVIEW. The driver should now appear in the palette. The driver palette allows you to browse the various VIs and functionality supported by the driver.



## Initialize the Instrument

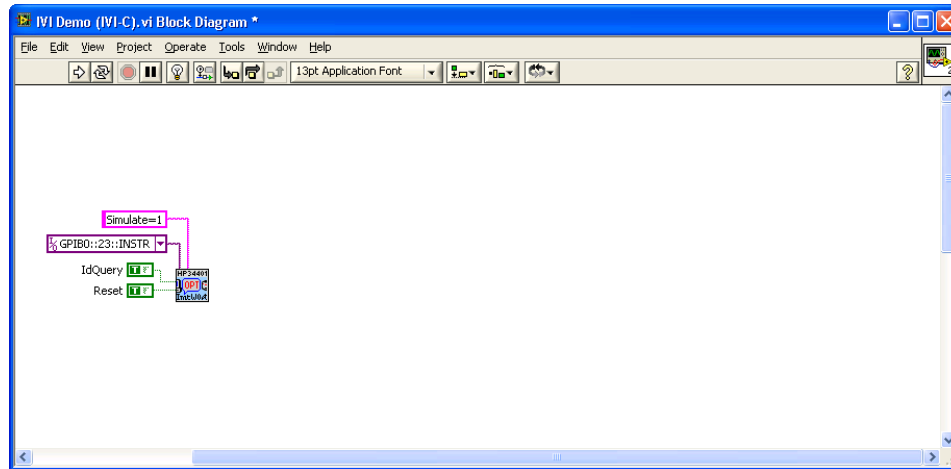
- 1 Select Initialize With Options VI from the hp34401a palette and place it on the Block Diagram.



- 2 Create constants and enter values for instrument resource name, ID Query, Reset, and IVI option string:
  - **GPIB0::23::INSTR** in the instrument resource name field
  - **True** in the ID Query field
  - **True** in the Reset field
  - **Simulate=1** in the Options field

**Note:** To create a constant, control, or indicator, right-click on the desired input terminal and select *Create*.

## Configure the Instrument



- 1 From the Configuration subpalette, select Configure Measurement VI and place it on the Block Diagram.
- 2 Create constants and enter values to set the resolution to 1 millivolt, the function to DC Voltage, and the range to 1.5 volts:
  - **0.001** in the Resolution field
  - **DC volts** in the Measurement Function field
  - **1.5** in the Range field
- 3 Connect the instrument handle and error terminals from Initialize With Options VI to Configure Measurement VI.
- 4 From the Trigger subpalette, select Configure Trigger VI and place it on the Block Diagram.
- 5 Connect resource name and error information from Configure Measurement VI to Configure Trigger VI.
- 6 Create a constant and enter a value of **0.01** in the Trigger Delay field.

**Note:** You can also set the Trigger Delay using a Property Node by replacing steps 4 & 5 with a property access as shown in the section “Setting a Property in an IVI-C Driver” below.

## Take the Reading

- 1 Return to the main hp34401a palette. From the Measurement subpalette, select Read VI and place it on the Block Diagram.
- 2 Set the value for Maximum Time to 1 second (1000 ms). Enter **1000** in the Maximum Time field.



- 3 Connect resource name and error information from Configure Trigger to Read VI.

## Display the Reading

Create an indicator for Reading from the terminal on the Read VI.

## Close the Session

- 1 Return to the main hp34401a palette. Select Close VI and place it on the Block Diagram.
- 2 Connect resource name and error information from Read VI to Close VI.  
**Note:** *LabVIEW compiles while developing, which lets you check the program execution at any time.*

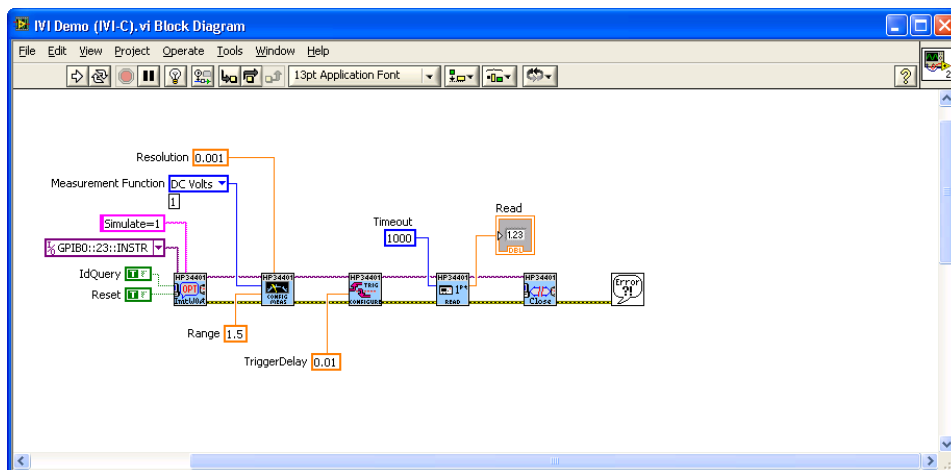
## Add Error Checking

- 1 Return to the main functions palette. From the Dialog & User Interface subpalette select Simple Error Handler VI and place it on the Block Diagram.
- 2 Connect the error information from Close VI to Simple Error Handler VI.

## Run the Application

Your final VI Block Diagram should contain the elements shown below. To run your VI:

- 1 Switch to the VI's Front Panel and click on the *Run* arrow to run the application.
- 2 The *Reading* indicator should display a simulated reading from the instrument.

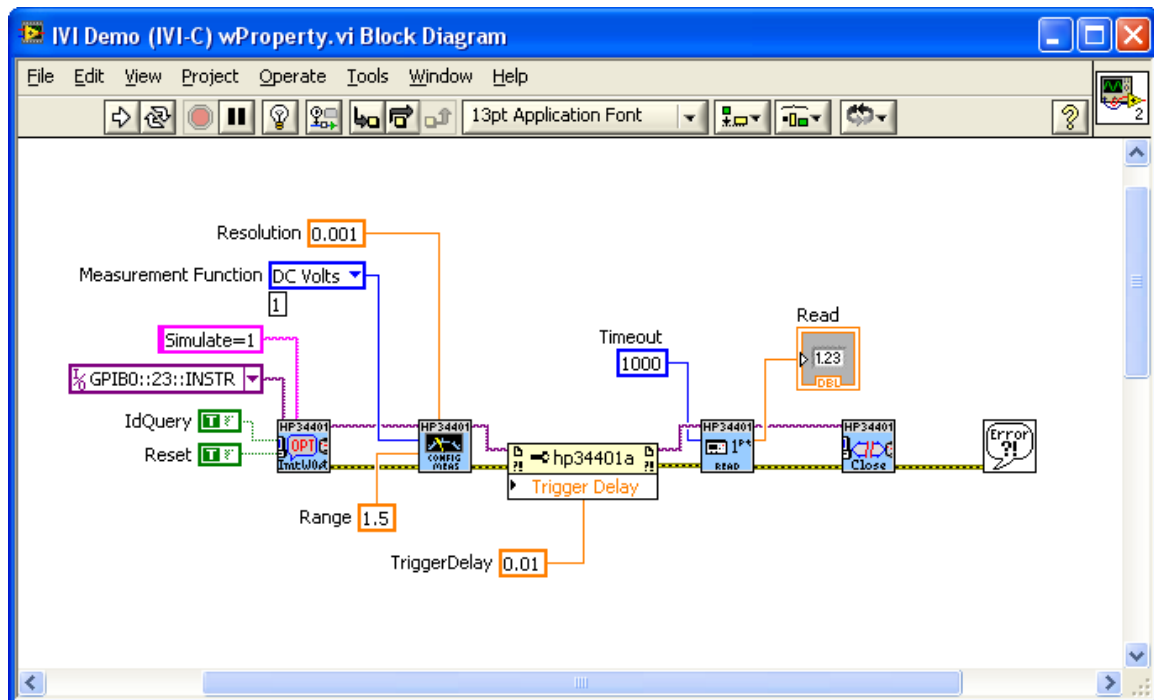


## Setting a Property in an IVI-C Driver

Properties such as Trigger Delay can also be set (and read) with a property node. This is important in cases where a configuration function is not provided by the driver.

For example we can replace steps 4 and 5 of the “Configure the Instrument” section with:

- 1 From the Functions palette select Application Control and drop a Property Node on the Block Diagram.
- 2 Connect the resource name and error information from Configure Measurement VI to the Property Node.
- 3 Right-click on the Property Node and select Change All to Write.
- 4 Click on the Property field and select Trigger >> Trigger Delay.



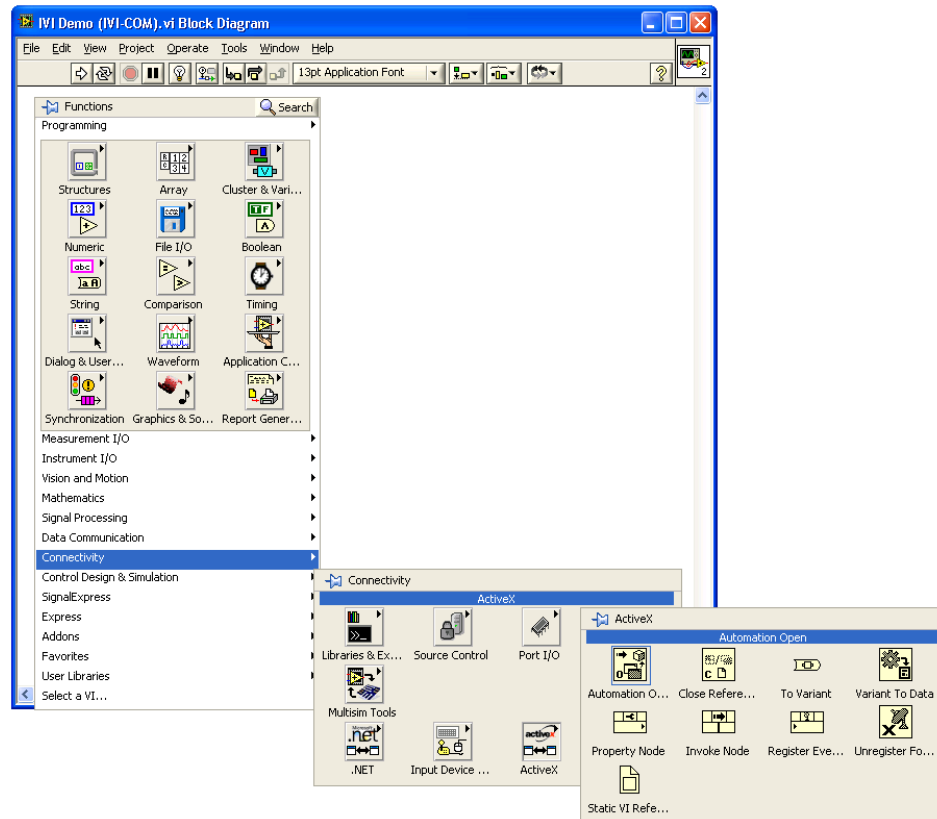
## Using IVI-COM

To use IVI-COM drivers in LabVIEW you will use the ActiveX functions and the Class Browser that are built-in to LabVIEW.

## Create a VI and Access the Driver

- 1 Launch LabVIEW.

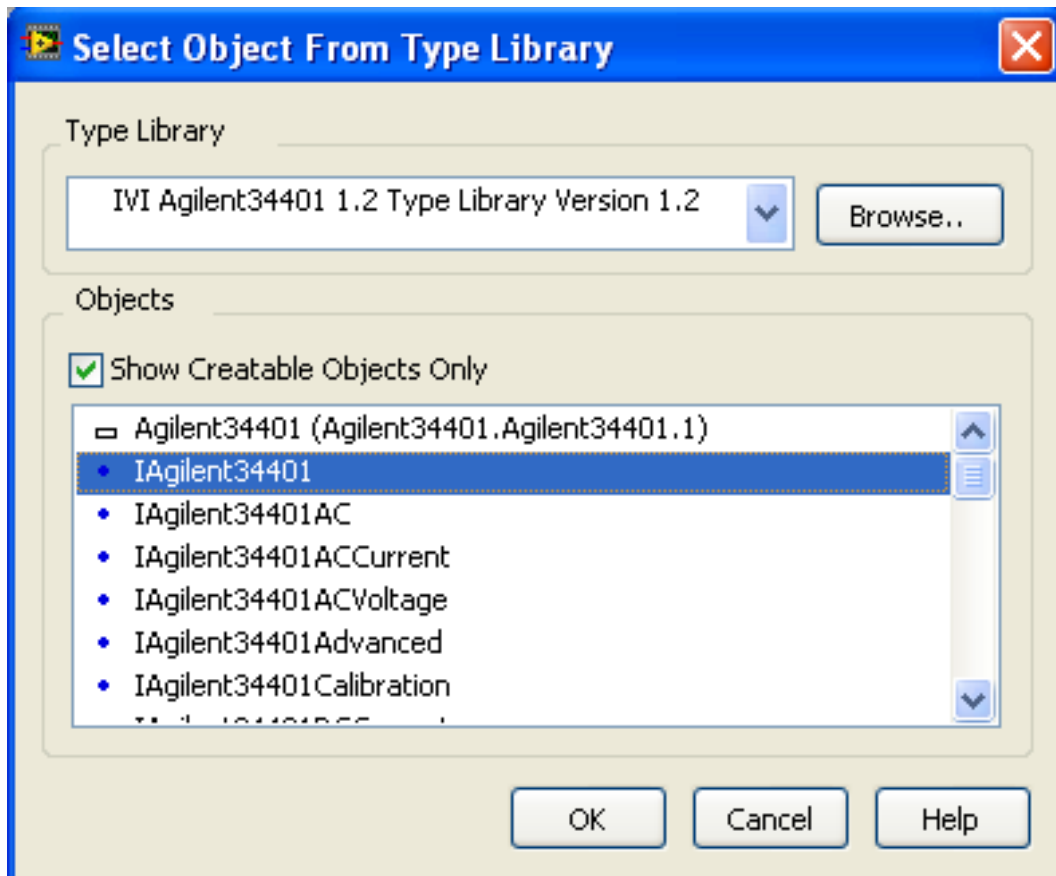
- 2 From the File menu, select New VI. The Front Panel and Block Diagram appear.
- 3 Right-click in the Block Diagram. The Functions palette appears.
- 4 Select the Connectivity subpalette and then the ActiveX subpalette. From this palette, you can access ActiveX and COM objects including all IVI-COM drivers.
- 5 Select Automation Open from the palette and place it on the block diagram.



- 6 Right-click on the Automation Refnum terminal, select Select ActiveX Class... and then Browse...

- 7 From the Type Library drop-down, select the IVI Agilent 34401A (Agilent Technologies) 1.2 Type Library Version 1.2, and then select the IAgilent34401 object. Click OK.

**Note:** If the IVI-COM driver you want to use is not listed, download and install the driver and close and restart LabVIEW. The driver should now appear in the type library browser.

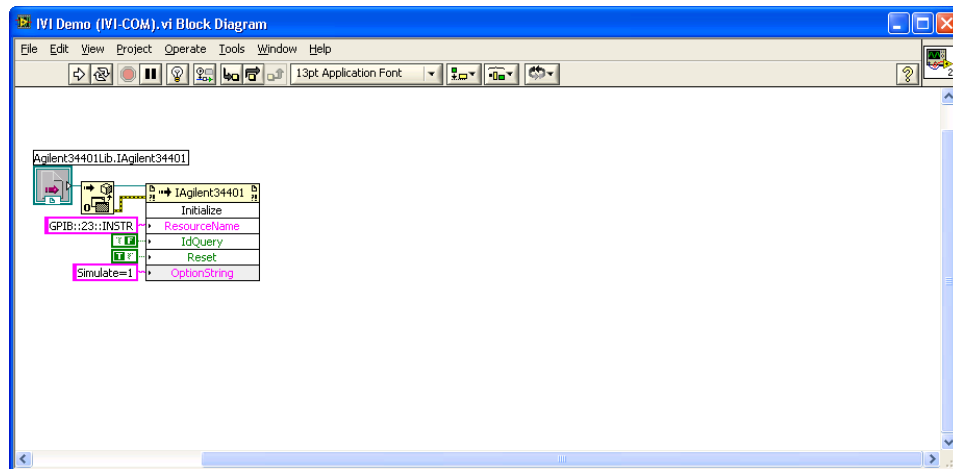


### Initialize the Instrument

- 1 From the View menu, select Class Browser. The Class Browser allows you to invoke methods and set or get properties of the ActiveX/COM object.
- 2 From the Object library drop-down, select ActiveX and then Select Type Libraries.
- 3 Scroll down and select the IVI Agilent 34401A (Agilent Technologies) 1.2 Type Library Version 1.2, Click OK.

- 4 Back in the Class Browser, under Properties and Methods, scroll down and select Initialize. Click Create and drag the Invoke Node to the Block Diagram.
- 5 Create constants and enter values for ResourceName, IDQuery, Reset, and OptionString:
  - **GPIB0::23::INSTR** in the instrument ResourceName field
  - **False** in the IDQuery field
  - **True** in the Reset field
  - **Simulate=1** in the OptionString field
- 6 Connect the automation refnum and error terminals from Automation Open to Initialize Invoke Node.

**Note:** Instead of using the Class Browser, you can select an Invoke Node from the ActiveX subpalette and select the Initialize method. To access driver properties, you can select a Property Node from the ActiveX subpalette and select the appropriate property or you can use the Class Browser for both IVI-C and IVI-COM drivers.



## Configure the Instrument

- 1 Go back to the Class Browser, and under Properties and Methods, double-click the DC Voltage property and select the Configure method. Click Create and drag the Invoke Node to the Block Diagram.
- 2 Create constants and enter values to set the Resolution to 1 millivolt and the Range to 1.5 volts:
  - **0.001** in the Resolution field
  - **1.5** in the Range field

- 3 Connect the automation refnum and error terminals from Initialize Invoke Node to DCVoltage.Configure Invoke Node.
- 4 In the Class Browser, go back to the top-level object and double-click the Trigger property and select the Delay property. Click Create Write and drag the Property Node to the Block Diagram.
- 5 Create a constant and enter a value of 0.01 seconds for the Delay field.
- 6 Connect the automation refnum and error terminals from DCVoltage.Configure Invoke Node to Trigger.Delay Property Node.

### Take the Reading

- 1 Return to the Class Browser, and under Properties and Methods, double-click the Measurement property and select the Read method. Click Create and drag the Invoke Node to the Block Diagram.
- 2 Set the value for Timeout to 1 second (1000 ms) by entering 1000 in the MaxTimeMilliseconds field.
- 3 Connect the automation refnum and error terminals from Trigger.Delay Property Node to Measurement.Read Invoke Node.

### Display the Reading

Create an indicator for Measurement.Read from the Invoke Node terminal.

### Close the Driver and Automation Sessions

- 1 Return to the Class Browser, and under Properties and Methods, double-click the Close method. Click Create and drag the Invoke Node to the Block Diagram.
- 2 Close the Class Browser. From the ActiveX subpalette, select Close Reference and place on the Block Diagram.
- 3 Connect the automation refnum and error terminals from Measurement.Read Invoke Node to Close Invoke Node and then to Close Reference function.

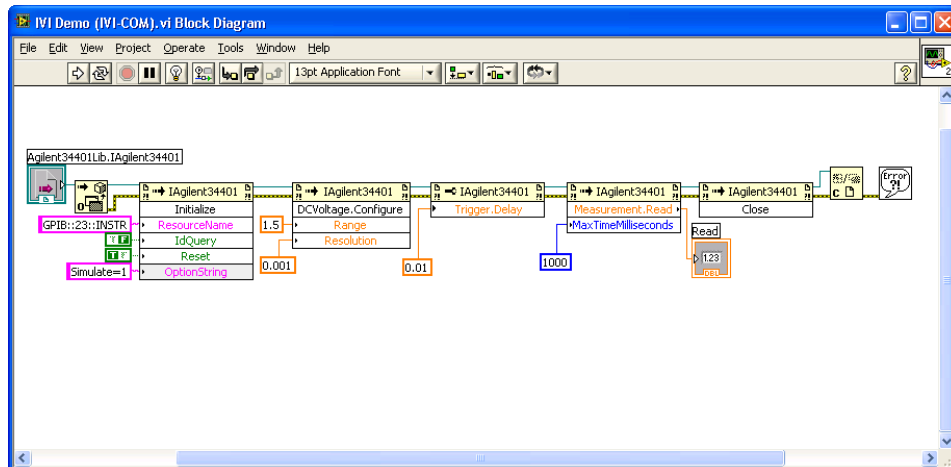
### Add Error Checking

- 1 Return to the main functions palette. From the Dialog & User Interface subpalette select Simple Error Handler VI and place it on the Block Diagram.
- 2 Connect the error information from Close Reference function to Simple Error Handler VI.

## Run the Application

Your final VI Block Diagram should contain the elements shown below. To run your VI:

- 1 Switch to the VI's Front Panel and click on the Run arrow to run the application.
- 2 The Reading indicator should display a simulated reading from the instrument.



## Further Information

Learn more about using an Minstrument driver in LabVIEW in this tutorial:  
<http://www.ni.com/tutorial/4556/en/>.