IVI FOUNDATION

Getting Started with IVI Drivers

Your Guide to Using IVI with Measure Foundry®

Version 1.1

Contents

Chapter 1
Introduction

Purpose

Welcome to ***Getting Started with IVI Drivers: Your Guide to Using IVI with Measure Foundry®***. 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 VXI*plug&play* Systems Alliance created specifications for instrument drivers called VXI*plug&play* drivers. Unlike SCPI, VXI*plug&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 VXI*plug&play* drivers do not provide a common programming interface. That means programming a Keithley DMM using VXI*plug&play* still differs from programming an Agilent DMM. For example, the instrument driver interface for one may be ke2000_read while another may be hp34401_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 VXI*plug&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:

Class	IVI Driver
Digital multimeter (DMM)	IviDmm
Oscilloscope	IviScope
Arbitrary waveform/function generator	IviFgen
DC power supply	IviDCPwr
AC power supply	IviACPwr
Switch	IviSwtch
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 VXI*plug&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 1.2.2.0 for 34401A, October 2008 (from Agilent Technologies)** … or the **Agilent 34401A IVI-C driver, Version 4.4, July 2010 (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 **Agilent 34401A (IVI-COM) or HP 34401 (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 Measure Foundry®

The Environment

Measure Foundry is a visual software environment for creating test and measurement, control, and analysis applications. The design environment consists of forms and the foundry window. You choose components from the foundry window and drop them onto your form. Clicking on the components accesses property pages where you can set design, configuration, and connectivity. This enables fast application development.

Each application consists of three primary elements:

- a data source supplies data to the application
- a control source determines how and when the data is used
- a data sink receives data to process or display

Example Requirements

- Measure Foundry 5
- Agilent 34401A IVI-COM, Version 1.2.2.0, October 2008 (from Agilent Technologies)
- Agilent IO Libraries Suite 16

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. You can also refer to Chapter 1, Download and Install IVI Drivers.

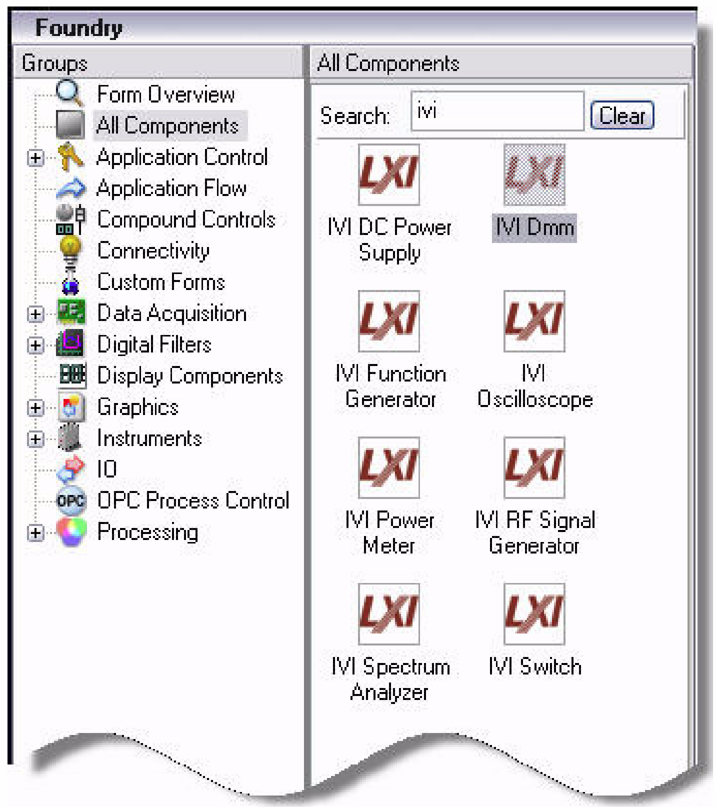
This example uses an IVI-COM driver. IVI-COM is the preferred driver for Measure Foundry. IVI-C is not supported.

Note*: You cannot specify an instrument in Measure Foundry unless you have already installed the appropriate driver. If you need to download and install a driver, you do not need to exit Measure Foundry. Install the driver and it appears in the list of available devices.*

Data Source

The property pages contain all the information necessary to create an instance of the driver, initialize and configure the instrument, set the trigger delay and reading timeout, and close the instrument.

1. Launch Measure Foundry.
2. Select File and Click New. The New Project screen opens.
3. In the Foundry window, select All Components. Enter IVI in the Search field.
4. Select the LXI IVI Dmm component, drag it to the Form, and drop it.



5. Double-click the LXI component. The Properties dialog appears.

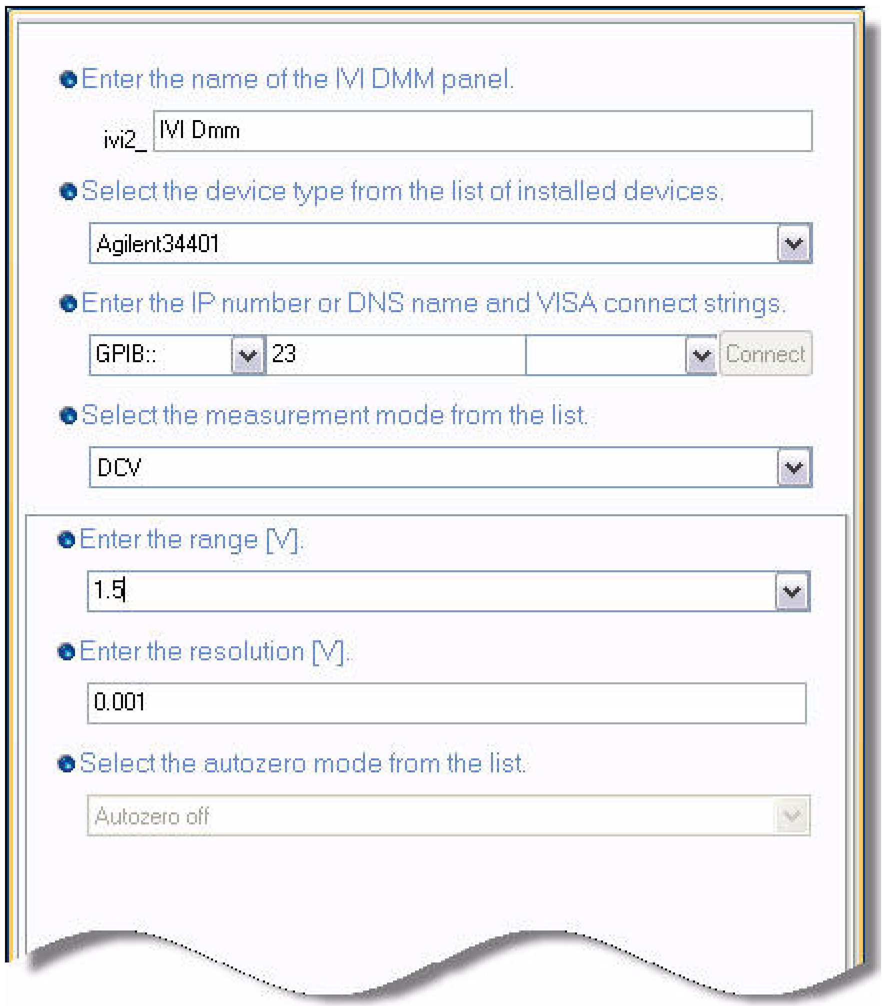
   Note*: The Properties dialog lets you access functions based on the purpose of your test.*

6. Configure the IVI DMM. To initialize the DMM and set the range to 1.5 volts and resolution to 1 millivolt, enter the following In the Properties dialog:

- **IVI Dmm** in the IVI DMM panel field
- **Agilent34401** in the device type field
- **GPIB** and **23** in the VISA connect string field
- **DCV** in the measurement mode field
- **1.5** in the range field
- **0.001** in the resolution field

Note: *Capabilities that are unavailable are grayed out.*

7 Click Next.

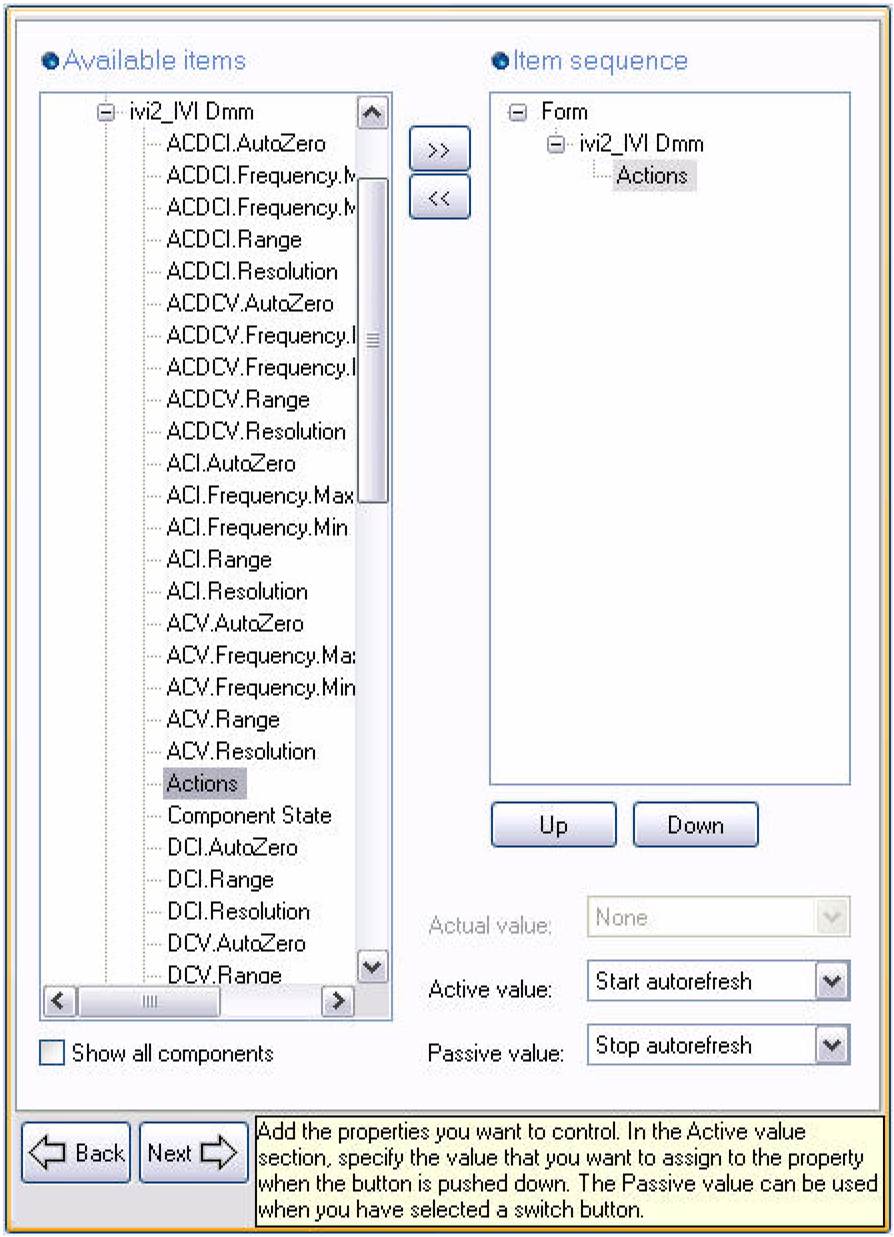


8 Configure the operating mode. Select **Auto refresh** and enter **1000** in the update rate field.
9 Click Next.
10 Configure the trigger delay. In the Properties dialog, enter **0.01** in the trigger delay field and **Immediate** in the trigger source field.
11 Click Next.
12 Configure the option string, reset, and reading timeout to 1 second. In the Properties dialog, enter the following:
   - **Simulate=true** in the IVI connect option string
   - **1000** in the timeout field
   - Check the Reset on Connect box
13 Click Next.

Control Source

The property pages contain the information necessary to label the button and specify how it controls the program.

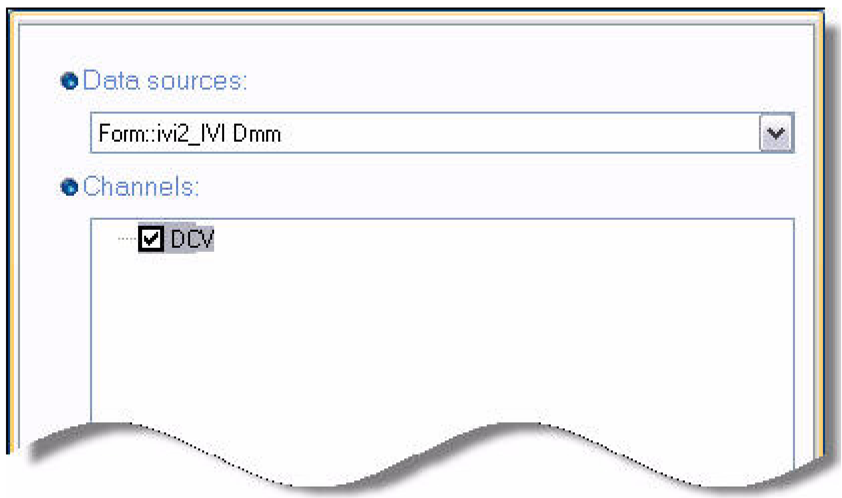
1 In the Foundry window, select Application Control. Drag and drop the Control Button to the form.
2 Double-click the Control Button. The Properties of Control Button dialog appears.
3 Enter **Start/Stop** in the text field.
4 Select the switch button type and click Next.
5 Scroll the list of Available items for the IVI Dmm and click Actions.
6 Select Actions and click the double arrow to add to Item sequence.
7 Select Start autorefresh from the drop-down list in Active value.
8 Select Stop autorefresh from the drop-down list in Passive value.


Available items
Item sequence
ivi2_IVI Dmm
ACDCI.AutoZero
ACDCI.Range
ACDCI.Resolution
ACDCV.AutoZero
ACDCV.Range
ACDCV.Resolution
ACI.AutoZero
ACI.Frequency.Max
ACI.Frequency.Min
ACI.Range
ACI.Resolution
ACV.AutoZero
ACV.Frequency.Min
ACV.Range
ACV.Resolution
Actions
Component State
DCI.AutoZero
DCI.Range
DCI.Resolution
DCV.AutoZero
DCV.Range
Form
ivi2_IVI Dmm
Actions
Up
Down
Actual value:
None
Active value:
Start autorefresh
Passive value:
Stop autorefresh
Show all components
Back
Next
Add the properties you want to control. In the Active value section, specify the value that you want to assign to the property when the button is pushed down. The Passive value can be used when you have selected a switch button.

Data Sink

The property pages contain all the information necessary to display the output.

1. In the Foundry window, select Display Components. Drag and drop the Single Value Label onto the form.
2. Double-click the Single Value Label. The Properties of Single Value Label dialog appears.
3. Scroll the list of Data sources and click Form:ivi2_IVI Dmm.
4. Check the box by DCV in the Channels field.



Compile and Run

The final program contains the IVC Dmm LXI component (data source), Start/Stop button (control source), and Single Value Label (data sink). From the Start Menu, click Compile and Run to check that your program runs.

Close Session

To close the session and release the driver, either exit the program or program a Control Button to set the Disconnect property to true in the IVI DMM component.

Further Information

Learn more about the Measure Foundry at [http://www.datatranslation.com/products/measurefoundry/.](http://www.datatranslation.com/products/measurefoundry/)

Measure Foundry® is a registered trademark of Data Translation® in the United States and/or other countries.