IVI Driver Toolset

User Manual

NATIONAL INSTRUMENTS™

June 2000 Edition
Part Number 321629B-01

Worldwide Technical Support and Product Information

`www.ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to `techpubs@ni.com`

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

CVI™, LabVIEW™, ni.com™, NI Spy™, PXI™ and VirtualBench™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Chapter 1
Introduction to IVI

Chapter 2
IVI Instrument Driver Overview

Chapter 3
IVI System Architecture

Chapter 4
Configuring Your System

Chapter 5
Class Driver Operation

Chapter 6
Advanced Class Driver Simulation

Appendix A
Technical Support Resources

Glossary

Index

Figures

Tables

About This Manual

This manual and associated online function/VI documents are intended for both LabWindows/CVI and LabVIEW users. The IVI Driver Toolset User Manual describes how to develop hardware independent test programs with IVI (Interchangeable *Virtual Instrument*) instrument drivers. This manual also gives an overview of IVI instrument drivers and the IVI system architecture, so that you can configure your system and develop test programs that are independent of your hardware. Follow the guidelines in this manual when you develop, debug, and deploy test programs that use IVI instrument drivers.

Conventions

The following conventions appear in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

✎ This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

`monospace italic` Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

1

Introduction to IVI

This chapter explains how to install the IVI Driver Toolset and describes what the setup program installs.

Installing IVI

To install the IVI Driver Toolset from CD, perform the following steps:

Note National Instruments recommends that you close other applications before you install the IVI Driver Toolset.

1. If you plan to use the IVI Driver Toolset with *LabVIEW* or *LabWindows/CVI*, ensure that you have already installed LabVIEW or LabWindows/CVI.
2. Insert your IVI Driver Toolset Installation CD into your CD-ROM drive.
3. Wait several seconds. Normally, the AutoRun utility on the CD automatically launches the IVI Driver Toolset Installation CD dialog box.
4. Follow the directions that appear on screen.

If the IVI Driver Toolset Installation CD dialog box does not appear, click the **Start»Run** in the Windows taskbar and enter *d*`:\setup` in the input box, where *d* is the letter of your CD-ROM drive. Click **OK** to begin installation.

What the Setup Program Installs

The setup program installs the following components on your hard disk.

IVI Class Drivers

The IVI Driver Toolset has five classes: oscilloscope, digital multimeter, function/arbitrary waveform generator, DC power supply, and switch. You use the IVI *class drivers* to develop hardware independent test programs.

The setup program installs the files necessary to use the class drivers with LabWindows/CVI and LabVIEW.

For LabWindows/CVI, an IVI class driver consists of the following files:

- The class driver program, which consists of a `.dll` file and import library (`.lib`) files for various compilers.
- The class driver *include file* (`.h`), which contains the function declarations and constant definitions for the class.
- The class driver *function panel* file (`.fp`), which contains information that defines the *function tree*, the function panels, and the help text.
- The `.sub` file, which documents attributes and their possible *values*.
- A Windows help file (`.hlp`), which contains documentation for the LabWindows/CVI class driver.

For LabVIEW, an IVI class driver consists of the following files:

- The class driver VIs, which are in a `.llb` file. The VIs link to a `.dll` file.
- The `.rc` file, which documents properties and their possible *values*.
- A set of `.mnu` files, which document the hierarchy of VIs for the class driver.
- A Windows help file (`.hlp`), which contains documentation for the LabVIEW class driver.

Advanced Class Simulation Tools

The advanced class simulation tools consist of a set of class *simulation drivers* and user interface resource (`.uir`) files. The class simulation drivers create simulation data for output parameters of class driver functions. The class simulation drivers can run in two modes: interactive and non-interactive. In the interactive mode, the class simulation drivers display panels that are in the `.uir` files. With these panels, you configure how the class simulation driver creates the output simulation data.

IVI Instrument-Specific Drivers

IVI instrument-specific drivers are software modules that contain functions and attributes for controlling a specific GPIB, RS-232, VXI, or PXI instrument.

IVI Engine

The IVI engine is a support library for the IVI class drivers and instrument-specific drivers.

VISA I/O Library

The VISA (Virtual Instrument Software Architecture) I/O Library gives instrument drivers a single interface for controlling GPIB, VXI, PXI, RS-232, or other types of instruments.

IVI Soft Front Panels

IVI soft front panels are applications that enable you to interactively operate your instruments. Use them to ensure that your instruments are operating correctly and to make simple, interactive measurements. Each class driver includes a soft front panel to control the instruments in your system. The soft front panel is a very helpful troubleshooting tool when you are debugging system issues or you want to take interactive measurements with your equipment. Because the soft front panels work with any instrument that plugs into the class drivers, you only have to learn how to use that one panel for all instruments of a given class.

NI Spy

NI Spy monitors Windows applications that use National Instruments drivers. NI Spy can monitor, record, and display calls made to IVI class drivers. Use NI Spy to quickly locate and analyze any erroneous calls that your application makes to the IVI class drivers.

IVI Configuration Utility

Use the *IVI Configuration utility* to configure your instrument-independent test system. With this utility, you create and configure IVI *logical names*. In your program, you pass logical names to one of the class driver initialize functions to identify the instruments and *specific drivers* to use.

2

IVI Instrument Driver Overview

This chapter introduces the concept of *instrument drivers* and explains how they have evolved. This chapter then introduces the concept of IVI instrument-specific and class drivers.

What Is an Instrument Driver?

In the early days of computer-controlled instrumentation systems, programmers used BASIC I/O statements in their application programs to send and receive command and data strings to and from the various instruments connected to their computer through the GPIB. Each instrument responded to particular ASCII strings as documented in each vendor’s instrument user manuals. Programmers were responsible for learning each command set and writing the control program.

Programming is often the most time-consuming part of developing an automated test system, especially when programmers have to learn the various command sets of the different instruments they use. This process requires even more time and resources when programmers need to create new applications with the same instruments. It became clear that programmers could save much time and money if they wrote high-level routines that hid the low-level commands. Also, if the routines were generic and modular, programmers could reuse them in any future application that used the same instrument. These reusable routines became known as instrument drivers.

Historical Evolution of Instrument Drivers

Although the concept of the instrument driver had promise, early implementations had serious limitations. Some approaches were too closely linked to proprietary development tools. Other approaches were too difficult to develop or modify. Users wanted drivers that were open and modifiable, built around standards that allowed instruments from a variety of vendors to peacefully coexist in one application.

The VXI*plug&play* systems alliance improved existing instrument driver standards. The VXI*plug&play* instrument driver architecture leveraged existing popular technology by building on the successful LabWindows/CVI and LabVIEW instrument driver standards.

These standards enable system interoperability. That is, you can install VXI*plug&play* instrument drivers from a variety of vendors on the same system without encountering system conflicts. In addition, these standards use VISA-defined data types to define parameters of all instrument driver functions. These data types promote the portability of instrument drivers to new operating systems and programming languages. Although the VXI*plug&play* model continues to represent a powerful instrument driver solution, it lacks several crucial features, such as, instrument interchangeability, execution performance, and test development flexibility.

The IVI Foundation

The IVI Foundation is an organization of end-users, instrument vendors, and system integrators who share a common commitment to promote the success of test system developers through open, powerful, instrument control technology. The IVI Foundation has extended the VXI*plug&play* instrument driver standards to incorporate features such as instrument interchangeability, execution performance, and test development flexibility. The IVI model achieves these advances without introducing additional complexity or performance overhead. Although IVI instrument drivers comply with the VXI*plug&play* standard, they have many additional features. The following items are some of the most important features:

- **Hardware Independence**—IVI class instrument drivers allow developers to build systems that reuse their test programs with different instruments. The benefits of instrument interchangeability extend to a wide variety of applications. The following are a few examples:
    - Test system developers in the military and aerospace industries, who must maintain test systems and code for many years, can easily reuse their test code on new equipment as instruments improve or become obsolete.
    - Manufacturers in competitive, high-volume industries, such as telecommunications and consumer electronics, can keep their production lines running when instruments malfunction or must be recalibrated.

  - Large manufacturing companies of all kinds can more easily reuse and share test code between departments and facilities without being forced to use the same instrumentation hardware.
- **Instrument State Caching**— Standard VXI*plug&play* drivers do not keep track of the state of the instrument. Therefore, each measurement function sets up the instrument for the measurement even if the instrument is already configured correctly. IVI drivers automatically cache the current state of the instrument. An IVI instrument driver function performs instrument I/O only when the instrument settings are different from what the function requires. This seemingly minor difference in approach leads to significant reductions in test time and cost.
- **Instrument Simulation**—IVI drivers can simulate the operation of an instrument when that instrument is not available. If you enable simulation, IVI drivers do not perform instrument I/O. Instead, IVI drivers range-check all input parameters and create simulated data for output parameters. With simulated data, developers can develop stable code for instruments even when the instruments are not available.

For a comprehensive list of IVI features, see Chapter 3, *IVI System Architecture*.

IVI Instrument-Specific Drivers

An instrument driver is a high-level function library that you use to control a specific GPIB, VXI, or serial instruments or other devices. With an instrument driver, you can easily control an instrument without knowing the low-level command syntax or I/O protocol.

The IVI instrument-specific drivers contain the information to control a particular instrument model, including the command strings, parsing code, and valid ranges of each setting for that particular instrument. IVI instrument drivers apply an attribute-based approach to instrument control to deliver better run-time performance and more flexible instrument driver operation.

In the remainder of this manual the term *specific driver* refers to an IVI instrument-specific driver. A specific driver gives you the following benefits over a traditional instrument driver:

- **State-caching**—State-caching eliminates redundant commands being sent to the instrument. If you try to set an attribute to a value that it already has, the specific driver skips the command.

- **Configurable range-checking**—Range-checking verifies that a value you specify for an attribute is within the valid range for the attribute. You can disable this feature for faster execution speed.
- **Configurable status query**—The status query feature automatically checks the status of the instrument after each operation. You can disable this feature for faster execution speed.
- **Simple Simulation**—You can develop application code that uses an instrument driver even when the instrument is not available. When in simulation mode, the instrument driver range-checks input parameters and generates simulated data for output parameters.
- **Multithread safety**—You can use IVI instrument driver in multithreaded applications. Multiple execution threads can use the same IVI instrument session without interfering with each other.

Instrument Classes—Standard Instrument Programming Interfaces

Interchangeability using IVI instrument-specific drivers is achieved through generic instrument class drivers. A class driver is a set of functions and attributes for controlling an instrument within a specified class, such as an oscilloscope, DMM, or function generator. The IVI Driver Toolset has five classes: oscilloscope, DMM, arbitrary waveform/function generator, switch, and DC power supply. Each one of these class drivers makes calls to IVI instrument-specific drivers to control the actual instruments. Through configuration, you can swap control to another instrument.

From your test program, you make calls to the class drivers, which in turn communicate through the specific drivers for your instruments. You can change the specific instrument drivers (and corresponding instruments) in your system underneath the class driver and your test code continues to work.

Figure 2-1 shows how a class driver redirects function calls from a test program to the correct specific instrument driver.

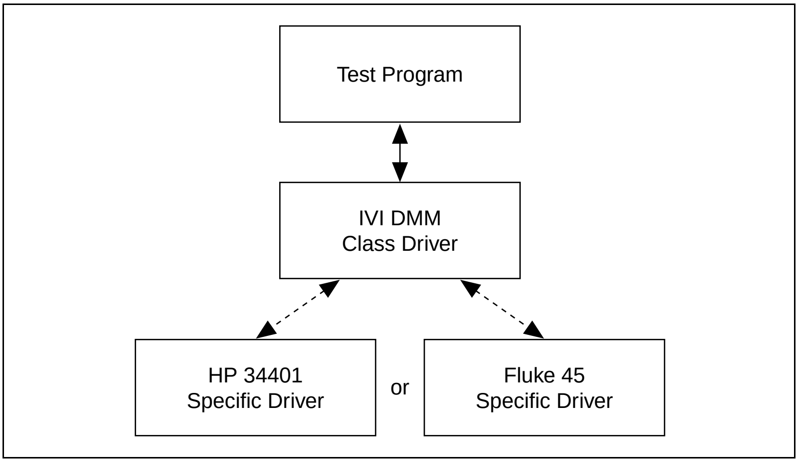


Figure 2-1. IVI Driver Architecture

The IVI Foundation manages the definition of the instrument classes. The IVI Foundation's charter is to define flexible programming interfaces for instrument classes that meet the needs of test system developers. The IVI Foundation has created specifications for common instrument classes. The IVI Foundation specifications define a standard Application Programming Interface (API) for each instrument class.

The National Instruments IVI class drivers conform to the IVI Foundation specifications. For example, the oscilloscope class contains a collection of attributes that are common to all oscilloscopes, such as vertical range, offset, timebase, trigger mode, and so on. The class also contains functions that set these attributes or retrieve data from the instrument, such as `ConfigureChannel`, `ConfigureAcquisitionRecord`, `ReadWaveform`, and so on. IVI class specifications give a standard definition for each of these functions and attributes for an oscilloscope. Programmers use these specifications to write test programs that work with any oscilloscope.

A Word on Interchangeability

Instrument interchangeability has long been a goal of many engineers who build test systems, particularly in the military and avionics industries. It is important for IVI users to remember that the interchangeability of IVI class drivers depends on the fundamental interchangeability of the hardware that you are using. The requirements of your test system are still the driving force behind your choice of particular instruments. For example, your test system requires DMM measurements with 8-1/2 digits of precision, you must use an 8-1/2 digit DMM. You cannot replace an 8-1/2 digit DMM with a 5-1/2 digit DMM in your test system, unless you require only 5-1/2 digits of precision, regardless of the software architecture. The IVI Driver Toolset implements a standard architecture for swapping instruments that are capable of delivering the measurement requirements of your test system.

Using IVI Instrument Drivers

In general you operate and develop test programs with IVI instrument drivers in the same way as with traditional instrument drivers. If you are using LabWindows/CVI, refer to the *LabWindows/CVI User Manual* for information on how to use instrument drivers with LabWindows/CVI. If you want information about using instrument drivers in LabVIEW, access the Index tab of *LabVIEW Online Reference* and go to the *Instrument Driver* index topic and select the *Getting Started* subtopic.

A specific driver consists of the following files:

- The instrument driver program, which consists of a source file (`.c`), a `.dll` file, and an import library (`.lib`) file.
- The instrument driver include (`.h`) file, which contains function declarations and constant definitions.
- The instrument driver function panel file (`.fp`), which contains information that defines the function tree, the function panels, and the help text.
- The `.sub` file, which documents attributes and their possible values.
- The `.llb` file which contains a set of LabVIEW VIs.
- The `.rc` file, which documents properties and their possible values.
- A set of `.mnu` files, which document the hierarchy of VIs.

The source code for a specific driver is the `.c` file. LabWindows/CVI is optimized for the creation and modification of specific drivers. Use the `.fp` and `.sub` files when you use an IVI instrument driver in the LabWindows/CVI environment.

Use the `.llb`, `.rc`, and set of `.mnu` files when you use an IVI driver in the LabVIEW environment. The VIs in the `.llb` file call the corresponding functions in the instrument driver `.dll` file. Therefore, an IVI driver in LabVIEW does not have G source code like a traditional LabVIEW driver.

LabVIEW instrument drivers previously shipped with G source code, and LabVIEW programmers used the G source code to debug or optimize the drivers. IVI drivers significantly reduce the necessity for instrument driver source code, are of significantly higher quality than traditional drivers and use state caching to optimize performance. The operation of IVI drivers is highly configurable. You can optimize the operation of an IVI driver without modifying the driver source code.

A single source approach for IVI drivers has many advantages for instrument driver users:

- Driver developers can produce instrument drivers faster to cover more instruments in your system.
- The rigorous internal structure of IVI drivers results in higher quality drivers than other existing models.
- National Instruments can more easily maintain and upgrade instrument drivers when each driver has only one set of source files. This results in higher quality instrument drivers for you to choose from when integrating a system.
- Consistency between LabVIEW and LabWindows/CVI instrument drivers means that it is easier for you to develop and maintain test systems that use both LabVIEW and LabWindows/CVI.

3

IVI System Architecture

This chapter describes the basic components of the IVI system architecture. This chapter also describes the IVI class drivers in detail, including the relationship of class drivers to specific drivers.

Basic Components of an IVI System

Figure 3-1 shows the components of an IVI system. In general, to implement an automated test system, you develop a program that controls instruments. For each instrument model that you access, you use an IVI specific driver. The specific driver encapsulates all the information to control a particular instrument model.

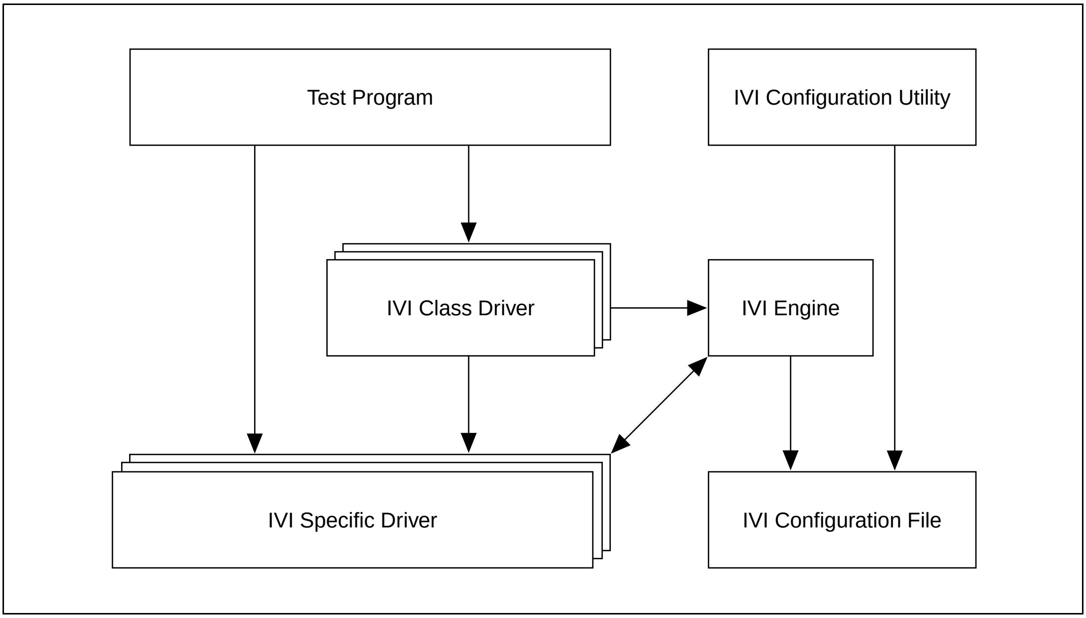


Figure 3-1. IVI System Architecture

Your test program communicates with the specific driver in one of two ways:

- **Directly**—Your test program can link directly to a specific driver. This is similar to the way in which you use traditional instrument drivers. With this approach, you gain most of the benefits of IVI instrument drivers, including state-caching, configurable range-checking, configurable status query checking, simple simulation, and multithread safety. All functions and attributes that the specific driver exports begin with a prefix that uniquely identifies the specific driver. Because the prefix is unique, you must modify your test program when you want to use a different instrument model or driver.
- **Through an IVI Class Driver**—Your test program can access a specific driver indirectly with an IVI class driver. Therefore, you get all the IVI benefits that reside with the specific driver.

  When you use an IVI class driver, you initialize the instrument by calling the initialize function in the IVI class driver. At this point, you pass a logical name to the initialize function. A logical name is a string that identifies the specific driver and the physical instrument to use. You create and configure logical names with the IVI Configuration utility.

  To change the instrument that your program uses, you edit the logical name to identify the new specific driver and physical instrument. You do not have to change or recompile your program. Therefore, the class driver allows you to develop hardware-independent test programs that do not require modification when you use a different instrument. For a complete description of the IVI Configuration utility refer to Chapter 4, *Configuring Your System*.

The preceding bulleted items show interchangeability as a benefit that you gain by using an IVI class driver. Interchangeability means that, whenever you want to change the instrument that your program uses, you edit the logical name to identify the new specific driver and physical instrument. You do not have to change or recompile your program. Therefore, the class driver allows you to develop hardware-independent test programs that do not require modification when you use a different instrument. For a complete description of the IVI Configuration utility refer to Chapter 4, *Configuring Your System*.

IVI class drivers permit you to develop test programs that are independent of specific hardware. A class driver is a set of functions and attributes for controlling an instrument within a specified class, such as an oscilloscope, DMM, or function generator. The IVI Driver Toolset has five classes: oscilloscope, DMM, arbitrary waveform/function generator, switch, and DC power supply.

In addition to interchangeability, IVI class drivers deliver other benefits such as advanced simulation, spying, and interchangeability checking. This section describes the features that are common to all IVI class drivers. For information regarding a particular IVI class driver, refer to the *IVI Class Driver Online Help*.

The remainder of this chapter discusses the general features of IVI class drivers in detail.

Class Driver APIs

The IVI class APIs conform to the specifications of the IVI Foundation.

Class Driver Prefix

The IVI class drivers work with a large set of specific drivers. Each class driver has a unique *class prefix* that gives the class driver unique and meaningful names and avoids conflicts with other instrument driver functions, attributes, and files. Each function name and attribute ID in the class driver begins with the class prefix. Also, the names of all component files (`.fp`, `.h`, `.dll`, `.llb`, and so on) of the class driver begin with the class prefix. Table 3-1 show the function, attribute ID, and filename prefixes for each IVI class driver.

Table 3-1. IVI Class Prefixes

IVI Class	Function Prefix	Attribute ID Prefix	Filename Prefix
Digital Multimeter	`IviDmm`	`IVIDMM_`	`ividmm.*`
Oscilloscope	`IviScope`	`IVISCOPE_`	`iviscope.*`
Function Generator	`IviFgen`	`IVIFGEN_`	`ivifgen.*`
DC Power Supply	`IviDCPwr`	`IVIDCPWR_`	`ividcpwr.*`
Switch	`IviSwtch`	`IVISWTCH_`	`iviswtch.*`

The following table shows proper use of the class prefix for the IVI class driver for digital multimeters.

Table 3-2. Proper Use of IVI Class Prefixes

Filename Prefixes (`ividmm.*`)	**Function Name Prefixes** (`IviDmm_`)	**Attribute IDs Prefixes** (`IVIDMM_`)
`ividmm.fp`, `ividmm.h`, `ividmm.dll`, `ividmm.lib`, `ividmm.sub`, `ividmm.llb`, `ividmm.rc`, and `ividmm*.mnu`	For example, `IviDmm_ConfigureMeasurement`, `IviDmm_Initiate`, and `IviDmm_Fetch`	For example, `IVIDMM_ATTR_FUNCTION`, `IVIDMM_ATTR_RANGE`, and `IVIDMM_ATTR_TRIGGER_SOURCE`

Note In LabVIEW, the VI names have blank spaces in place of the underscores, for example, IviDmm Configure Measurement, IviDmm Initiate, and IviDmm Fetch.

Class Capability Groups

Because many instruments of a given class have additional, differing functionality or capability, no single programming interface can work with all instruments. For this reason, the IVI class drivers divide the instrument capabilities into *Inherent IVI Capabilities, Base Capabilities, Extension Groups*, and *Instrument-Specific Capabilities*.

Inherent IVI Capabilities

Inherent IVI Capabilities are the functions and attributes that all IVI class drivers implement. Table 3-3 shows the inherent IVI functions. For a complete description of the IVI inherent functions refer to the *IVI Class Driver Online Help*.

Note To complete the function names in the following table for any one of the IVI class drivers, replace *`ClassPrefix`* with one of the class prefixes: `IviDmm`, `IviDCPwr`, `IviFgen`, `IviScope`, or `IviSwtch`. For example, for the digital multimeter class *ClassPrefix* Initialize VI becomes IviDmm Initialize VI in LabVIEW, and *`ClassPrefix`*`_init` becomes `IviDMM_init` in LabWindows/CVI.

Table 3-3. Inherent IVI VIs/Functions

LabVIEW Inherent VI	LabWindows/CVI Inherent Function
Initialize/Close Category	

ClassPrefix Initialize
ClassPrefix Initialize With Options
ClassPrefix Close
	`ClassPrefix_init`
	`ClassPrefix_InitWithOptions`
	`ClassPrefix_close`

Set, Get, and Check Attribute/Property Category	

Use Property Node VI to get and set attributes, which is the standard LabVIEW approach.	`ClassPrefix_SetAttribute<type>`
	`ClassPrefix_GetAttribute<type>`
	`ClassPrefix_CheckAttribute<type>`

Utility Category	

ClassPrefix Reset
ClassPrefix Self-Test
ClassPrefix Revision Query
ClassPrefix Error-Query
ClassPrefix Error Message
	`ClassPrefix_reset`
	`ClassPrefix_self_test`
	`ClassPrefix_revision_query`
	`ClassPrefix_error_query`
	`ClassPrefix_error_message`

Error Info Subcategory	
Use the General Error Handler VI to view errors, which is the standard LabVIEW approach.	`ClassPrefix_GetErrorInfo` `ClassPrefix_ClearErrorInfo`
Interchangeability Info Subcategory	

ClassPrefix Get Next Interchange Warning
ClassPrefix Reset Interchange Check
ClassPrefix Clear Interchange Warnings
	`ClassPrefix_GetNextInterchangeWarning`
	`ClassPrefix_ResetInterchangeCheck`
	`ClassPrefix_ClearInterchangeWarnings`

Coercion Info Subcategory	
ClassPrefix Get Next Coercion Record	`ClassPrefix_GetNextCoercionRecord`
Locking Subcategory	
You cannot lock IVI sessions in LabVIEW.	`ClassPrefix_LockSession` `ClassPrefix_UnlockSession`

Note You use VIs and properties when you configure IVI drivers in LabVIEW. You use functions and attributes when you configure IVI drivers in LabWindows/CVI.

The table of IVI inherent VIs and functions contains the following major sections:

- **Initialize/Close**—Initialize and close instrument driver sessions.
- **Attribute Access**—Set, get, and check the values of attributes. Type-safe functions exist for each attribute data type. The possible data types are `ViInt32`, `ViReal64`, `ViString`, `ViBoolean`, and

`ViSession`. You insert one of these values in place of the place holder, `<type>`.

- **Utility**—Control common instrument operations. These functions include many of the functions that VXI*plug&play* requires, such as reset, self-test, revision query, error query, and error message. This class also contains functions that access IVI error information, access interchangeability warnings, access coercion records, and lock the instrument driver session.

Table 3-4 shows the inherent IVI attributes. For a complete description of the inherent IVI attributes refer to the *IVI Class Driver Online Help*.

Table 3-4. Inherent IVI Properties/Attributes

LabVIEW Property	LabWindows/CVI Attribute
User Options Category	
ClassPrefix Range Check	`CLASSPREFIX_ATTR_RANGE_CHECK`
ClassPrefix Query Instrument Status	`CLASSPREFIX_ATTR_QUERY_INSTR_STATUS`
ClassPrefix Cache	`CLASSPREFIX_ATTR_CACHE`
ClassPrefix Simulate	`CLASSPREFIX_ATTR_SIMULATE`
ClassPrefix Use Specific Simulation	`CLASSPREFIX_ATTR_USE_SPECIFIC_SIMULATION`
ClassPrefix Record Value Coercions	`CLASSPREFIX_ATTR_RECORD_COERCIONS`
ClassPrefix Interchange Check	`CLASSPREFIX_ATTR_INTERCHANGE_CHECK`
ClassPrefix Spy	`CLASSPREFIX_ATTR_SPY`
Class Driver Identification Category	
ClassPrefix Description	`CLASSPREFIX_ATTR_CLASS_DRIVER_DESCRIPTION`
ClassPrefix Class Driver Prefix	`CLASSPREFIX_ATTR_CLASS_DRIVER_PREFIX`
ClassPrefix Class Driver Vendor	`CLASSPREFIX_ATTR_CLASS_DRIVER_VENDOR`
ClassPrefix Major Version	`CLASSPREFIX_ATTR_CLASS_DRIVER_MAJOR_VERSION`
ClassPrefix Minor Version	`CLASSPREFIX_ATTR_CLASS_DRIVER_MINOR_VERSION`
ClassPrefix Revision	`CLASSPREFIX_ATTR_CLASS_DRIVER_REVISION`
ClassPrefix Class Specification Major Version	`CLASSPREFIX_ATTR_CLASS_DRIVER_CLASS_SPEC_MAJOR_VERSION`
ClassPrefix Class Specification Minor Version	`CLASSPREFIX_ATTR_CLASS_DRIVER_CLASS_SPEC_MINOR_VERSION`

Table 3-4. Inherent IVI Properties/Attributes (Continued)

LabVIEW Property	LabWindows/CVI Attribute
Specific Driver Identification Category	
ClassPrefix Description	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_DESCRIPTION
ClassPrefix Specific Driver Prefix	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_PREFIX
ClassPrefix Specific Driver Locator	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_LOCATOR
ClassPrefix Specific Driver Vendor	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_VENDOR
ClassPrefix Major Version	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_MAJOR_VERSION
ClassPrefix Minor Version	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_MINOR_VERSION
ClassPrefix Revision	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_REVISION
ClassPrefix Class Specification Major Version	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION
ClassPrefix Class Specification Minor Version	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION
Specific Driver Capabilities Category	
ClassPrefix Supported Instrument Models	*CLASSPREFIX*_ATTR_SUPPORTED_INSTRUMENT_MODELS
ClassPrefix Group Capabilities	*CLASSPREFIX*_ATTR_GROUP_CAPABILITIES
ClassPrefix Function Capabilities	*CLASSPREFIX*_ATTR_FUNCTION_CAPABILITIES
ClassPrefix Attribute Capabilities	*CLASSPREFIX*_ATTR_ATTRIBUTE_CAPABILITIES
ClassPrefix Number of Channels	*CLASSPREFIX*_ATTR_NUM_CHANNELS
Instrument Identification	
ClassPrefix Manufacturer	*CLASSPREFIX*_ATTR_INSTRUMENT_MANUFACTURER
ClassPrefix Model	*CLASSPREFIX*_ATTR_INSTRUMENT_MODEL
ClassPrefix Firmware Revision	*CLASSPREFIX*_ATTR_INSTRUMENT_FIRMWARE_REVISION
Error Info Category	
Use the General Error Handler VI to view errors, which is the standard LabVIEW approach	*CLASSPREFIX*_ATTR_PRIMARY ERROR
Use the General Error Handler VI to view errors, which is the standard LabVIEW approach	*CLASSPREFIX*_ATTR_SECONDARY ERROR
Use the General Error Handler VI to view errors, which is the standard LabVIEW approach	*CLASSPREFIX*_ATTR_ERROR_ELABORATION

Table 3-4. Inherent IVI Properties/Attributes (Continued)

LabVIEW Property	LabWindows/CVI Attribute
Advanced Session Information Category	
ClassPrefix Logical Name	`CLASSPREFIX_ATTR_LOGICAL_NAME`
ClassPrefix Resource Descriptor	`CLASSPREFIX_ATTR_RESOURCE_DESCRIPTOR`
You cannot get session properties from LabVIEW property node..	`CLASSPREFIX_ATTR_IO_SESSION_TYPE`
You cannot get session properties from LabVIEW property node.	`CLASSPREFIX_ATTR_IO_SESSION`

Note You use VIs and properties when you configure IVI drivers in LabVIEW. You use functions and attributes when you configure IVI drivers in LabWindows/CVI.

The attributes table for IVI inherent attributes contains the following sections:

- **User Options**—Contains attributes that the user can set to affect the behavior of class drivers and specific drivers.
- **Class Driver Identification**—Contains attributes that describe the version, supplier and prefix of the class driver.
- **Specific Driver Identification**—Contains attributes that describe the version, supplier, location and prefix of the specific driver.
- **Specific Driver Capabilities**—Contains attributes that describe various capabilities of the instrument that are implemented in the specific driver.
- **Instrument Identification**—Contains attributes that provide model and firmware information about the instrument controlled via the instrument driver.
- **Error Info**—Contains attributes for reporting and retrieving error information.
- **Advanced Session Information**—Contains attributes that provide resource and IO session information regarding the class and specific driver.

Base Capabilities

The base capabilities are the functions and attributes of an instrument class that are common to most of the instruments of that class. The goal is to cover 95 percent of the instruments in a particular class. For a specific driver to be compliant with a class, it must implement all the base capabilities.

For example, the base capabilities of the oscilloscope class have functions and attributes that configure an edge-triggered acquisition, initiate an acquisition, and return the acquired waveform. For a complete description of the base capabilities for a particular class, refer to the *IVI Class Driver Online Help*.

Extension Groups

Extension groups are groups of functions and attributes that represent more specialized features of an instrument class that are not common to all instruments of that type. Specific instrument drivers are not required to implement extension groups. With extension groups, the IVI class drivers create standard programming interfaces for features and capabilities that are *not* standard in every instrument of that class.

For example, although all oscilloscopes have very similar base capabilities for vertical and horizontal settings, there is a wide variety of trigger modes among oscilloscopes. The IviScope class driver includes extensions for different trigger modes, such as TV trigger, runt trigger, width trigger, and so on. Every scope that has TV triggering can comply with the TV trigger extension group functions and attributes of the IviScope class. However, an oscilloscope that does not implement the TV trigger extension group still can be compliant with the IviScope class, because the oscilloscope can support the base capabilities of the IVI Scope class.

After you use an extension group in your program, any instrument you use with the program must implement the extension group. The online help file, the LabWindows/CVI function panels, and the LabVIEW VIs for the class drivers mark functions and attributes that are a part of an extension group with a special symbol. For example, the online help for the IviScope class driver marks the functions and attributes that control the TV trigger extension group with the symbol `[TV]`. For a complete description of the extension groups of a particular class refer to the *IVI Class Driver Online Help*.

Instrument-Specific Capabilities

IVI specific drivers can export not only inherent IVI capabilities, base instrument capabilities, and extension groups, but also whatever instrument-specific functions and attributes that the driver developer considers necessary to control the features of a device that are not interchangeable with other devices.

Only the specific driver implements the instrument-specific capabilities. Therefore, if you want to use instrument-specific capabilities in your test program, you must access the specific driver directly. Refer to the online help of a specific driver for documentation regarding the instrument-specific features of that instrument driver.

Your test program can access both the class driver and the specific driver using the same instrument driver session. To do so, you must initialize the instrument session by invoking the initialize function in the class driver. You use the session that the class driver initialize function returns when you invoke functions in the either the class driver or the specific driver. The ability to call both class driver and specific driver functions with the same IVI session allows you to develop test programs that are mostly interchangeable and access instrument-specific capabilities only when necessary. You must modify the portions of your test program that access instrument-specific capabilities for your program to work with a different instrument.

When you initialize an instrument through a class driver, you can specify default values for instrument-specific attributes. The IVI default setup feature allows you to set the initial values of the instrument-specific attributes, and also helps you to protect interchangeability.

Class Driver Relationship to Instrument-Specific Drivers

Figure 3-2 illustrates an example of the relationship between the capability groups of a class driver and a specific driver. In this example, the class driver implements the inherent IVI capabilities, the base instrument capabilities, and two extension capability groups. The specific driver implements the inherent IVI capabilities, the base instrument capabilities, one of the extension capability groups, and instrument-specific capabilities.

The class driver dynamically loads the specific driver. The IVI class drivers require that the specific driver be in a `.dll`, `.c`, `.obj`, or `.lib` file. The specific drivers normally come in `.dll` and `.c` file formats. If you use the class drivers outside of the LabWindows/CVI environment, you *must* use the `.dll` file. If you are using the class drivers within the LabWindows/CVI environment, it is usually best to use the `.dll` file.

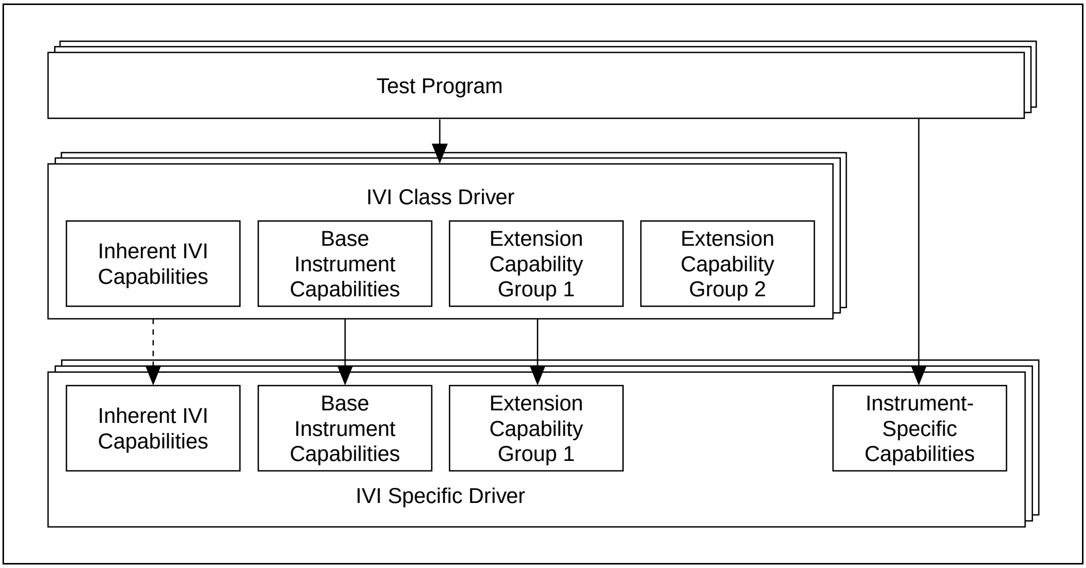

Test Program
IVI Class Driver
Inherent IVI Capabilities
Base Instrument Capabilities
Extension Capability Group 1
Extension Capability Group 2
Inherent IVI Capabilities
Base Instrument Capabilities
Extension Capability Group 1
Instrument-Specific Capabilities
IVI Specific Driver

Figure 3-2. Relationship of the Class Driver and Specific Driver

In general, when a test program calls functions and attributes of a capability group in the class driver, the class driver maps these calls to the corresponding capability group in the specific driver. If the test program attempts to call extension group functions that the specific driver does not implement, the class driver returns an error.

The class driver does not implement instrument-specific capabilities. Instead, the test program accesses the instrument-specific capabilities by invoking functions and attribute of the specific driver directly.

Your test program can access the inherent IVI capabilities, base instrument capabilities, and extension capabilities by invoking functions and attributes in either the class or specific driver. However, to achieve interchangeability it is best to use the class driver.

Note To access the capabilities of the instrument through the class driver, you must create the instrument driver session by calling the initialize function in the class driver.

In Figure 3-2, a dashed line connects the inherent IVI capabilities of the class driver and specific driver because some inherent IVI functions and attributes are accessible only through the class driver. If you initialize an instrument driver session by calling the initialize function in the specific driver, you cannot access the class-only functions and attributes.

You can access the IVI interchangeability functions—Get Next Interchange Warning, Reset Interchange Check, and Clear Interchange Warning—only through the class driver.

The following table lists the inherent IVI attributes that you can access only through the class driver.

Note To complete the attribute names in the following table for any one of the IVI class drivers, you replace *CLASSPREFIX* with one of the class prefixes: IVIDMM, IVIDCPWR, IVIFGEN, IVISCOPE, or IVISWTCH. For example, in the digital multimeter class *CLASSPREFIX*_ATTR_SPY becomes IVIDMM_ATTR_SPY.

Table 3-5. Inherent IVI Attributes That You Access Only through the Class Driver

Description	Function Name
Interchangeability Check	*CLASSPREFIX*_ATTR_INTERCHANGE_CHECK
Specific Driver Module Pathname	*CLASSPREFIX*_ATTR_SPECIFIC_DRIVER_LOCATOR
Class Type	*CLASSPREFIX*_ATTR_CLASS_TYPE
Class Major Version	*CLASSPREFIX*_ATTR_CLASS_DRIVER_MAJOR_VERSION
Class Minor Version	*CLASSPREFIX*_ATTR_CLASS_DRIVER_MINOR_VERSION
Class Revision	*CLASSPREFIX*_ATTR_CLASS_DRIVER_REVISION
Class Specification Major Version	*CLASSPREFIX*_ATTR_CLASS_DRIVER_SPEC_MAJOR_VERSION
Class Specification Minor Version	*CLASSPREFIX*_ATTR_CLASS_DRIVER_SPEC_MINOR_VERSION
Class Driver Vendor	*CLASSPREFIX*_ATTR_CLASS_DRIVER_VENDOR
Class Driver Description	*CLASSPREFIX*_ATTR_CLASS_DRIVER_DESCRIPTION
Spy	*CLASSPREFIX*_ATTR_SPY
Use Specific Simulation	*CLASSPREFIX*_ATTR_USE_SPECIFIC_SIMULATION

Table 3-5. Inherent IVI Attributes That You Access Only through the Class Driver (Continued)

Description	Function Name
Attribute Capabilities	*CLASSPREFIX*_ATTR_ATTRIBUTE_CAPABILITIES
Function Capabilities	*CLASSPREFIX*_ATTR_FUNCTION_CAPABILITIES

Default Setup

For some instruments, you might have to configure instrument-specific capabilities so that the instrument behaves in an interchangeable manner. However, when you configure instrument-specific capabilities in your program, you significantly reduce the likelihood that your program can work with another instrument. The IVI default setup feature allows you to set the initial values of instrument-specific capabilities, without sacrificing interchangeability.

When you initialize an instrument through a class driver, you can specify default values for instrument-specific attributes. You can also specify the default values for attributes with the IVI Configuration utility. The class driver applies the default setup as part of the initialization routine. With the default setup mechanism you can configure instrument-specific attributes external to your program.

This feature is also useful for extension group attributes. An example is the IviDmm extension attribute that sets the power line frequency. Most DMMs must know the power line frequency so that they can correctly calculate aperture times based on power line cycles (PLCs). Typically, GPIB DMMs have a built-in power supply and directly connect to the power line. Each DMM can sense the power line frequency or else it has a thumbwheel switch that you use to physically configure the power line frequency. For these DMMs you cannot programmatically configure the power line frequency, and the specific drivers for these instruments do not implement the power line frequency attribute.

Alternatively, plug-in DMMs in a PXI, PCI, VXI, or PCMCIA form factor do not directly connect to the power line. They receive power from the computer or chassis in which they are installed. You must programmatically configure the power line frequency for these DMMs to operate. However, to configure the power line frequency attribute in your program requires that any DMM that you use with the program implement the power line frequency attribute. Thus, to maximize the number of DMMs you can use, you configure the power line frequency attribute through the default setup mechanism. If the next DMM you use with the program does not implement the power line frequency attribute, you remove the power line frequency from the default setup. Thus, your program can work with DMMs that do or do not implement power line frequency attribute.

For information on how to configure a default setup, refer to Chapter 4, *Configuring Your System*.

Instrument Simulation

You can simulate your instrumentation hardware with IVI drivers. Using the simulation features, you can develop test code even when your instruments are not available. You can also use simulation to test for interchangeability. If you use IVI technology to build a test system, you can integrate the driver of the new instrument and run test programs against this driver in simulation mode. In effect, you can test a new instrument before you purchase it to make sure it can work in your system.

An IVI driver implements three different simulation capabilities:

- **Instrument driver calls**—With IVI simulation, you can make function calls from your test program to an instrument driver without each function returning an error. When you enable simulation, IVI instrument drivers do not attempt to communicate with the instrument. Therefore, you can make instrument driver function calls from your program and not receive I/O errors because the instrument is not present.

  It is important to notice that you can use this simulation capability with IVI class drivers as well as with specific drivers. Thus, you can simulate calls to a class driver and to instrument-specific functions in the specific driver.

  You can enable simulation with the IVI Configuration utility. Alternatively you can use the Initialize With Options function to enable simulation. The Initialize With Options function allows you to pass a string that presets a number of driver attributes. One of these attributes is the simulation attribute. When you use this function to preset the simulation attribute, you alert the driver that the instrument is not connected to the computer.
- **Range Checking and Parameter Coercion**—You can range check and coerce all input parameters. Every time you try to send a value to the instrument to configure a particular setting, the driver first ensures that the value is valid for the particular setting on the particular instrument model. If necessary, the specific driver coerces the value to

a value that is acceptable for the instrument. The range-checking and coercion operations happen completely within the specific driver software. Even when you write test code without connecting the instrument, the specific driver ensures that each value you are attempting to send to an instrument is valid.

Note The specific driver performs the range-checking and coercion operations. Therefore, you can use this simulation capability with a class driver or a specific driver.

- **Simulated Output Data**—You can create simulated data for output parameters of functions. The IVI architecture can use one of the following methods to generate simulated data:
  - **Specific Driver Simulation**—Every specific driver has basic built-in algorithms to simulate data generation. For example, when you use a DMM-specific driver in simulation mode, the Read function returns a random number within the valid range of the current mode of the DMM. When you use an oscilloscope-specific driver in simulation mode, the Read Waveform function returns a simple sine-wave array of data within valid ranges for the scope. This simple data generation process returns data values to the program variables so subsequent function calls do not fail for lack of data. However, a simple sine wave of fixed frequency might not be meaningful.
  - **Advanced Class Driver Simulation**—Because you have access to the specific driver source code, you are free to modify the driver. Therefore, you can add your own data generation algorithms to generate simulated data that more closely applies to the UUT or the application on which you are working. However, this code is useful only for that particular specific driver. If you change instruments in the future, you must re-implement this work for each new instrument you add. When you want customized simulation code, the IVI class drivers provide advanced simulation tools, which the following section describes.

Advanced Class Driver Simulation

The IVI class drivers provide advanced simulation tools that you can use to create simulated output data. The class drivers provide advanced simulation by using a *simulation driver*. Simulation drivers are “virtual instruments” that plug into a class driver. The class driver uses the simulation driver to generate data.

For example, the IVI Driver Toolset includes five simulation drivers, one for each class driver—oscilloscope, DMM, arbitrary waveform/function generator, switch, and DC power supply. Each of these simulation drivers plugs into the corresponding class driver to perform data generation that is more flexible than data generation in instrument-specific drivers. Figure 3-3 illustrates how a class driver uses a class simulation driver.

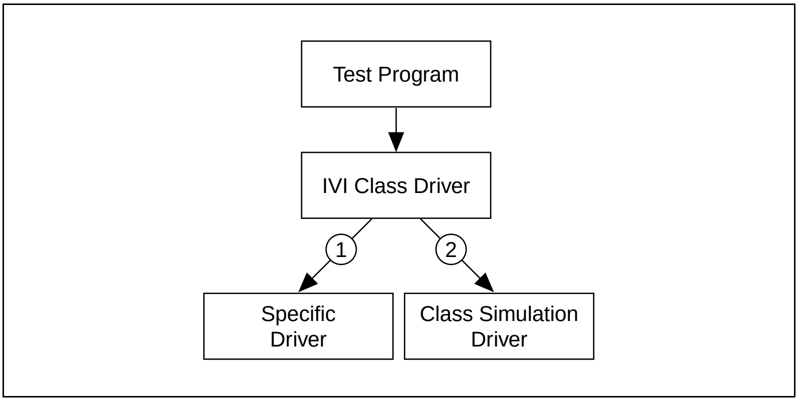
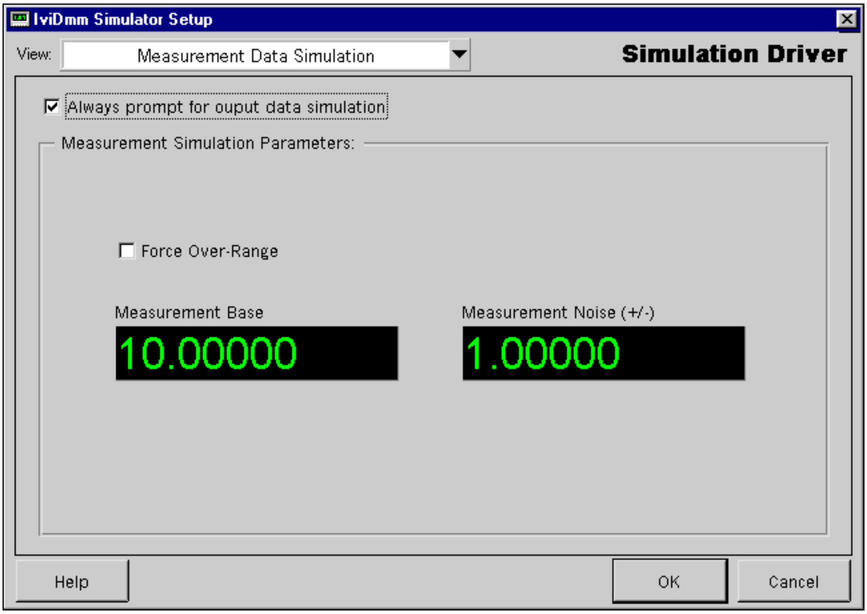


Figure 3-3. Class Simulation Driver

When your program makes a call to a class driver, the class driver calls the corresponding function in the specific driver. When you enable simulation, the class driver opens an additional session, a session that is assigned to its class simulation driver. Whenever you invoke a function in the class driver, the class driver first calls the corresponding function in the specific driver and then calls the same function in the class simulation driver. Under this mechanism, the specific driver still range checks and coerces all input parameters, but the class simulation driver generates the simulated output data and status codes.

Simulation drivers have two modes: interactive and non-interactive. You specify whether to use interactive or non-interactive simulation with the IVI Configuration utility. In interactive mode, simulation drivers have pop-up user interface panels that allow you to configure the parameters for generating the simulated output data.

For example, when you initialize the IviDmm class driver in simulation mode, the simulation driver displays the panel shown in Figure 3-4.



View: Measurement Data Simulation

☑ Always prompt for ouput data simulation

Measurement Simulation Parameters:

□ Force Over-Range

Measurement Base: 10.00000

Measurement Noise (+/-): 1.00000

Figure 3-4. IviDmm Simulation Driver User Interface

From the panel, you can select a base measurement value and an offset. For example, in Figure 3-4, the panel settings specify a value of 10.0 with a range of ±1.0. You can configure the driver to display the panel each time the program calls a function that returns measurement data, or you can configure the driver to generate the data automatically within the range you specify. In automatic mode, the simulation driver returns data to the program without requiring further user interaction.

In addition to generating simulated measurement data, you can use the advanced class simulation features to generate simulated results for the Self-Test, Error Query, and Revision Query functions that all IVI drivers export. You also can use the class simulation tools to generate simulated completion codes for the instrument driver functions. You use this feature to verify that your program correctly handles error conditions that the instrument driver might return.

To configure all the features of the simulation driver, you use attributes in the simulation driver. When you use non-interactive simulation, you configure the attributes of the simulation driver with the IVI Configuration utility. In this manner, you can configure how the simulation driver works without modifying your test program code.

The IVI Driver Toolset includes C source code for the class simulation drivers. You can develop very robust simulated data generation algorithms for your test systems and plug them into the simulation drivers. Because simulation drivers work with the class drivers, you can reuse simulation code that you have developed, whenever you swap to a different instrument.

Disabling Unused Extensions

You might develop test programs that do not use one or more of the extension groups that a class driver defines. Normally your program should work with instruments that do not implement the extension group as well as with those that do. However, if your program does not configure an extension group and the specific driver implements that extension group, the values of the attributes in the unused extension group are unknown. The attributes are likely to be set to the power-on settings of the device. The power-on settings often vary from instrument to instrument, so when you rely on a specific driver to implement an extension group, you might sacrifice interchangeability.

To accommodate instruments that implement extension groups that your program never configures, the class driver disables unused extensions by setting the extension groups to an interchangeable state. This interchangeable state for an extension group configures the extension group to have no effect on the behavior of the instrument.

For example, the IviDmm base capabilities control DMMs that can take a single measurement. The IviDmm class defines a multipoint extension group that controls DMMs that can acquire multiple samples from multiple triggers. If you develop a program that uses only the IviDmm base capabilities with an instrument that implements the multipoint extension group, the IviDmm class driver sets the multipoint extension group attributes to an interchangeable state when you call the Initiate or Read functions for IviDmm.

To disable the multipoint extension group to the interchangeable state, the IviDmm class driver sets the trigger count attribute to `1` and the sample count attribute to `1`. In this configuration, the multipoint extension group does not affect the instrument's behavior. Therefore, you can run the program with instruments that implement only the IviDmm base capabilities as well as with instruments that implement the multipoint extension group.

If your program has ever set any of the values of an extension group, the class driver does not configure the extension group. For information regarding the interchangeable state that the class drivers apply for disabling unused extensions, refer to the *IVI Class Driver Online Help*.

Interchangeability Checking

The IVI class drivers have a feature called interchangeability checking. Interchangeability checking verifies that your program will produce the same behavior when you use it with a different instrument.

You can enable interchangeability checking in one of the following ways:

- In the IVI Configuration Utility, select **Interchange Check** in the Inherent Attributes tab of the properties dialog box for your virtual instrument. Chapter 4, *Configuring Your System*, describes the properties dialog box.
- Set the value of the interchangeability checking attribute to `VI_TRUE` in your program.
- Set the interchangeability checking attribute to `VI_TRUE` in the option string parameter of the `InitWithOptions` function or Initialize With Options VI.

When you enable interchangeability checking, the class driver queues interchangeability warnings when it encounters instrument configurations that might not produce the same behavior when you use a different instrument. You use the NI Spy to view these warnings or call the Get Next Interchange Warning function.

Interchangeability Checking Rules

The following rules govern interchangeability checking in IVI.

An interchangeability check occurs when you call a class driver operation that depends on the current state of the instrument. For example, the IviDmm class driver performs interchangeability checking when your program calls any of the following functions.

Table 3-6. IviDmm Functions that Invoke Interchangeability Checking

Name in LabWindows/CVI	Name in LabVIEW
`IviDmm_Initiate`	IviDmm Initiate
`IviDmm_Read`	IviDmm Read
`IviDmm_ReadMultiPoint`	IviDmm Read MultiPoint

The class driver performs interchangeability checking on a capability group basis. When you enable interchangeability checking, the class driver always performs interchangeability checking on the base capabilities group. In addition, the class driver performs interchangeability checking on all extension groups for which you have set any of the attributes. If your program has never set any attributes of an extension group, the class driver does not perform interchangeability checking on that group.

In general, a class driver generates an interchangeability warning when it encounters one of the following conditions:

- **Unspecified State**—An attribute that affects the behavior of the instrument is in a state that you did not specify. This can happen if your program does not configure the attribute or if your program configures the attribute but the value becomes invalid as a result of your program configuring a different attribute.

  If an attribute is in a state that you did not specify, then the value of the attribute is either a power-on setting of the instrument or a value that the instrument sets based on the configuration of another attribute. In either case, the value of the attribute, and therefore the behavior of the instrument, is likely to be different when you run the program with a different instrument.

- **Instrument-Specific Value**—You set a class attribute to an instrument-specific value. Many attributes that a class driver defines represent a set of discrete settings. For these attributes, the class driver defines the possible values to which you can set the attribute. Specific drivers can define additional, instrument-specific values for the attribute. When your program sets an attribute to an instrument-specific value, it is likely to behave differently when you run it with different instruments.

For example, the attribute that configures the measurement function in the IviDmm class can have both class values and instrument-specific values. The class driver defines values for common measurement functions such as AC volts, DC volts, AC current, DC current, and others. Specific drivers can define instrument-specific values for the attribute. The constant values that one specific driver uses can overlap with the constant values that other specific drivers use. One specific driver might define an instrument-specific measurement function and another specific driver might use the same value to define an entirely different measurement function. Therefore, using an instrument-specific value in your program can result in different measurement results depending on which instrument you use.

- **Read-Only Attribute**—You configure the value of an attribute that the class defines as read only. In a few cases the class drivers define read only attributes that specific drivers might implement as read/write.

  The attributes of the IviDmm class that return the aperture time are examples of this interchangeability issue. With some DMMs, you can set the aperture time as well as read it. The specific drivers for these DMMs might implement the attributes for the aperture time as read/write. Most specific drivers that are compliant with the class implement the attribute as read only. Therefore, if your program configures an attribute that a class defines as read-only, it is likely that your program will not work with other instruments.

- **Value Not Configured**—The class driver encounters an error when it tries to apply a value to an extension attribute that your program never configures. The *Disabling Unused Extensions* section earlier in this chapter describes how the class driver sets the attributes of extension groups for which you have never specified values. It sets the attributes to an interchangeable state. The purpose is to make your program behave the same regardless of whether the instruments you use implement the extension group. Some instruments that implement the extension group might not support the value to which the class driver attempts to set the attribute. In this case, the class driver queues an interchangeability warning instead of returning an error from the function.

  A good example is the attribute that configures the interpolation method in the IviScope class. If your program never sets that value of this attribute and the specific driver implements the attribute, the

IviScope class driver attempts to set the interpolation method to `Sin(x)/x`. However, some oscilloscopes always interpolate data points. For these cases, the class driver generates an interchangeability warning to indicate that the attribute that controls the interpolation method is not in an interchangeable state.

Each IVI class driver defines exceptions to the interchangeability checking rules and defines which functions perform interchangeability checking. Refer to the *IVI Class Driver Online Help* for the interchangeability checking rules for a particular class.

Viewing Interchangeability Warnings

If you set the *`CLASSPREFIX`*`_ATTR_SPY` attribute to `VI_TRUE`, you can use the NI Spy utility to view interchangeability warnings. The NI Spy utility highlights functions in blue that have interchangeability warnings. Chapter 5, *Class Driver Operation*, describes how to use the NI Spy utility to view interchangeability warnings.

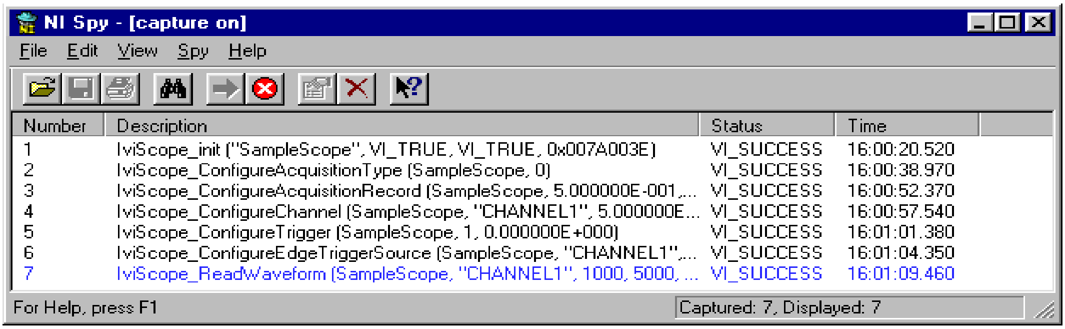
If you set the *`CLASSPREFIX`*`_ATTR_SPY` attribute to `VI_FALSE`, you can use the Get Next Interchange Warning function to retrieve interchangeability warnings programmatically. You can set the value of the *`CLASSPREFIX`*`_ATTR_SPY` attribute with the IVI Configuration utility. Chapter 4, *Configuring Your System*, describes how to use the IVI Configuration utility.

Note You must retrieve interchange warnings if you set the *`CLASSPREFIX`*`_ATTR_SPY` attribute to `VI_FALSE`. Otherwise, the queue might build up and appear as a memory leak.

Spying

NI Spy is an application monitor for Windows applications using National Instruments drivers. It can monitor, record, and display calls made to IVI class drivers. You can use NI Spy to quickly locate and analyze any erroneous calls that your application makes to the IVI class drivers.

When you enable spying on a particular class driver session, NI Spy captures all function calls that your application makes on that class driver session. You enable spying on a particular class driver session with the IVI Configuration utility or by setting the *`CLASSPREFIX`*`_ATTR_SPY` attribute to `VI_TRUE`. Figure 3-5 shows a sample trace from the NI Spy utility.



Number	Description	Status	Time
1	IviScope_init ("SampleScope", VI_TRUE, VI_TRUE, 0x007A003E)	VI_SUCCESS	16:00:20.520
2	IviScope_ConfigureAcquisitionType (SampleScope, 0)	VI_SUCCESS	16:00:38.970
3	IviScope_ConfigureAcquisitionRecord (SampleScope, 5.000000E-001,...	VI_SUCCESS	16:00:52.370
4	IviScope_ConfigureChannel (SampleScope, "CHANNEL1", 5.000000E...	VI_SUCCESS	16:00:57.540
5	IviScope_ConfigureTrigger (SampleScope, 1, 0.000000E+000)	VI_SUCCESS	16:01:01.380
6	IviScope_ConfigureEdgeTriggerSource (SampleScope, "CHANNEL1",...	VI_SUCCESS	16:01:04.350
7	IviScope_ReadWaveform (SampleScope, "CHANNEL1", 1000, 5000, ...	VI_SUCCESS	16:01:09.460

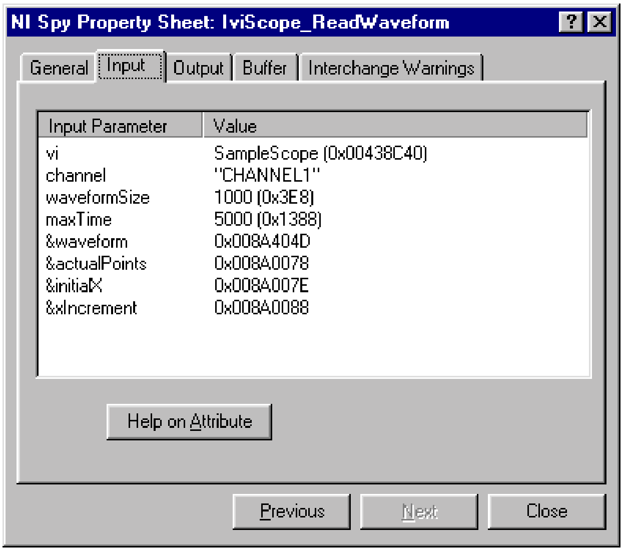
For Help, press F1

Captured: 7, Displayed: 7

Figure 3-5. Sample NI Spy Trace

NI Spy records all input parameters that you pass to a function and all output parameters that the function returns. NI Spy also displays the return value of the function.

You can see detailed information for every call NI Spy captures through property sheets. Figure 3-6 shows a sample property sheet.



Input Parameter	Value
vi	SampleScope (0x00438C40)
channel	"CHANNEL1"
waveformSize	1000 (0x3E8)
maxTime	5000 (0x1388)
&waveform	0x008A404D
&actualPoints	0x008A0078
&initialX	0x008A007E
&xIncrement	0x008A0088

Figure 3-6. Sample NI Spy Property Page

The property sheets display detailed information such as the following items:

- Information about the application that made each call and the time stamp of the call
- The input and output parameters to each call, and the contents of buffer parameters
- Descriptive error information
- Interchangeability warnings
- Information regarding the coercion of attribute values

For complete information regarding the use of the NI Spy utility to monitor calls to class drivers, refer to Chapter 5, *Class Driver Operation*.

4

Configuring Your System

This chapter describes how you use the IVI Configuration utility to create and configure logical names and virtual instruments for your system. This chapter also describes how you use the IVI Configuration utility to swap instruments without modifying, recompiling, or re-linking your application program. The IVI Configuration utility is a section within the National Instruments Measurement & Automation Explorer.

When you call an initialize function for a class driver, you pass a logical name that identifies a particular virtual instrument. The virtual instrument, in turn, identifies a particular physical instrument and specific driver and specifies the initial settings for the session. If you want to use your program with a different physical instrument, you change the configuration of the logical name to use the virtual instrument for the new physical instrument. You can change the initial settings for the session by changing the configuration of the virtual instrument.

Launching the IVI Configuration Utility

The IVI Configuration utility is integrated into your Windows operating system and appears in the Windows Explorer. Use one of the following methods to launch the IVI Configuration utility:

- Click the **Start** button in the Windows taskbar and select **Programs»National Instruments»IVI Driver Toolset»IVI Configuration**.
- Double-click the Measurement & Automation icon on the Windows desktop.

The *explorer* view of the IVI Configuration utility appears in Figure 4-1.

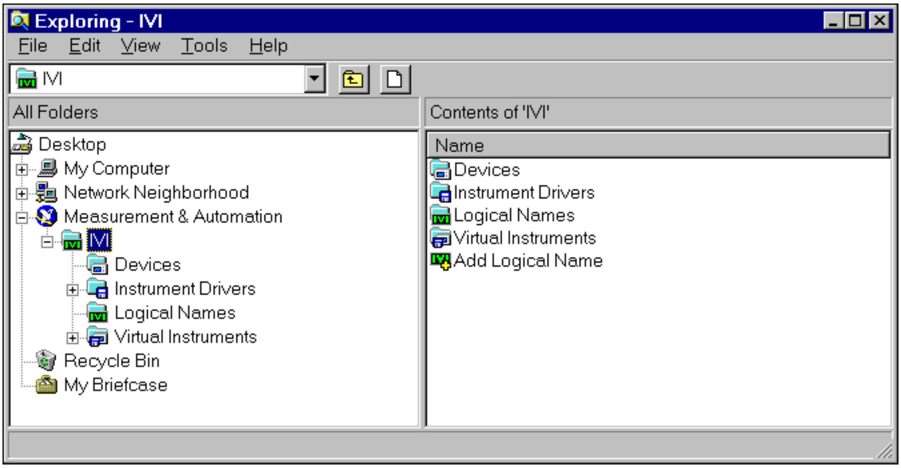


Figure 4-1. Explorer View of the IVI Configuration Utility

The explorer view of the IVI Configuration utility contains a *tree* window and a *list* window. The tree window displays a hierarchical list of the folders that you can configure. The list window displays the contents of the folder that you select in the tree window. To configure IVI logical names, open the `Measurement & Automation` folder and then open the `IVI` folder.

The *cabinet* view of the IVI Configuration utility appears in Figure 4-2.

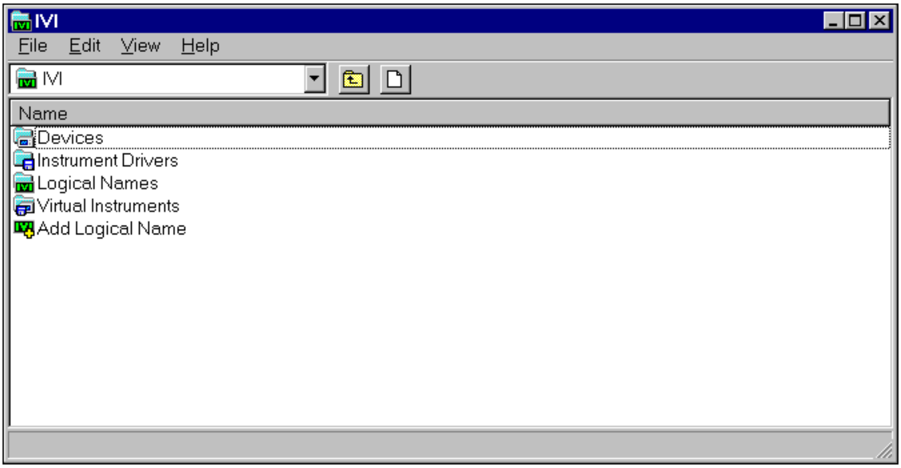


Figure 4-2. Cabinet View of the IVI Configuration Utility

The cabinet view of the IVI Configuration utility displays only the list window. To configure IVI logical names, open the `IVI` folder.

IVI Configuration Utility Overview

This section describes the features of the IVI Configuration utility.

IVI Configuration Utility Organization

The IVI folder contains all the items that you can configure for an IVI system. The items are grouped into subfolders.

The following table shows the hierarchy of folders in the IVI folder and the icon that the IVI Configuration utility uses to identify the type of item each folder contains.

Table 4-1. Organization of IVI Configuration Utility

Folder Name	Icon
IVI	none
Logical Names	IVI
Virtual Instruments	💾
Simulation Virtual Instruments	💾 S
Instrument Drivers	💾 IVI
Class Drivers	💾 C IVI
Simulation Drivers	💾 S IVI
Devices	IVI IVI IVI IVI

The IVI folder contains a wizard that helps you create new logical names. Refer to the *Creating a Logical Name* section in this chapter for an example of how to use the Logical Name Wizard. The following subfolders also appear in the IVI folder:

- **Logical Names**—Contains logical names that reference particular virtual instruments in the Virtual Instruments folder. If you want to swap an instrument, you edit the properties of the corresponding logical name to reference a new virtual instrument.

- **Virtual Instruments**—Contains virtual instruments that refer to specific drivers in the `Instrument Drivers` folder and to particular devices in the `Devices` folder. A virtual instrument also specifies initial settings for attributes.
  - **Simulation Virtual Instruments**—Contains *simulation virtual instruments* that refer to particular simulation drivers from the `Simulation Drivers` folder and to the initial attribute settings for those drivers.
- **Instrument Drivers**—Contains IVI specific driver configuration items that specify a code module, such as a dynamic link library, for a specific driver. The class driver uses the code module to communicate with the device.
  - **Class Drivers**—Contains IVI Class driver configuration items that refer to a particular default simulation virtual instruments. The class driver uses the default simulation virtual instrument when you enable simulation, unless the virtual instrument for the session specifies another simulation virtual instrument.
  - **Simulation Drivers**—Contains IVI simulation driver configuration items that specify code modules, such as a dynamic link library, that a class driver can use to simulate a device.
- **Devices**—Contains IVI device configuration items that specify the address of a device to use.

Editing IVI Configuration Items

In general, you use the IVI Configuration utility in the same way that you use the Windows Explorer. You can select, cut, copy, paste, delete, and rename items within each folder just as you do with files. For example, to rename an item, first select the item in the Explorer view. Then click once on the selected item to begin editing the name.

To create a new item, open the folder that contains the type of item you want to create and perform one of the following actions:

- Right-click on the folder and select the **Insert** command from the menu that appears.
- Click the **Insert** button in the toolbar.

Use the properties dialog box to edit the properties of an item. To invoke the Properties dialog box, perform one of the following actions:

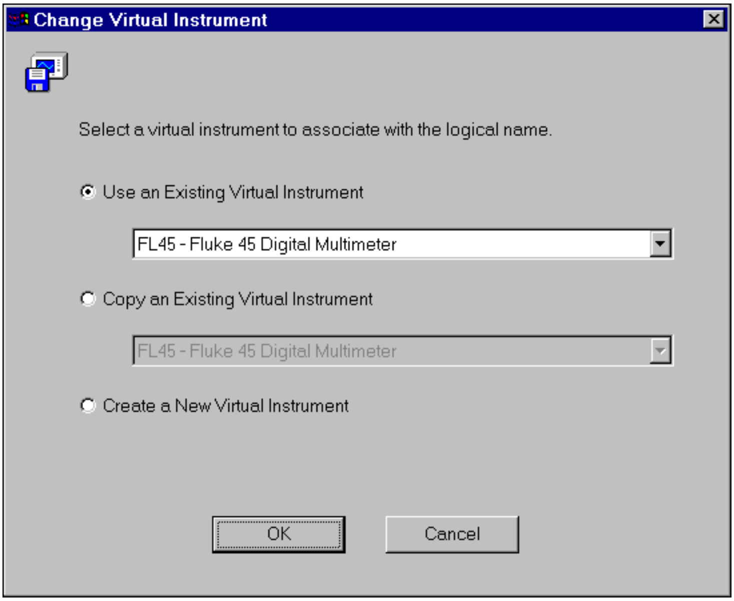
- Double-click an item.
- Right-click the item and select the **Properties** command from the context menu.

Changing Items that Another Item References

Many IVI configuration items refer to other items by name. In these cases, the referring item is the *parent* and the item it refers to is the *child*. For example, each logical name—a parent—refers to a virtual instrument—a child—and each virtual instrument refers to a device and a specific driver, which are children. Notice in these examples that a parent item, such as a virtual instrument, can also be a child item.

Logical names, virtual instruments, class drivers, and simulation virtual instruments are IVI configuration items that are parents; they can refer to other items. The Properties dialog boxes for these parent items display the names of the child items to which they refer. A **Change** button and a **Properties** button appear next to the name of each child item. Use the **Change** button to specify a different child item. Use the **Properties** button to modify the properties of the current child item.

To change the child item to which the parent item refers, click the **Change** button. A generic Change *Item* dialog box appears with a title that is specific to the type of the child item. For example, if you click the **Change** button of the Properties dialog box for a logical name, the Change Virtual Instrument dialog box appears, as shown in Figure 4-3.



Change Virtual Instrument

Select a virtual instrument to associate with the logical name.

● Use an Existing Virtual Instrument

FL45 - Fluke 45 Digital Multimeter

○ Copy an Existing Virtual Instrument

FL45 - Fluke 45 Digital Multimeter

○ Create a New Virtual Instrument

Figure 4-3. Change Virtual Instrument Dialog Box

Each Change *Item* dialog box allows you to choose a new child item. You can use an existing item, copy an existing item, or create a new item. To use an existing item, select the **Use an Existing *Item*** option, and choose an item from the ring control below the option. Click the **OK** button to return to the original Properties dialog box.

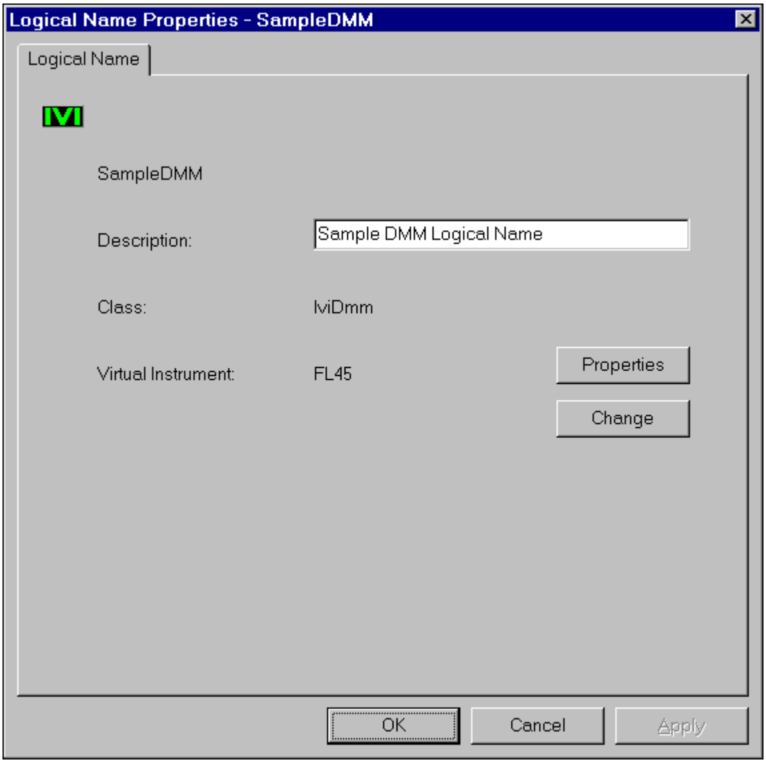
To copy an existing item, select the **Copy an Existing *Item*** option and select an item to copy from the ring control below the option. To create a new item, select the **Create a New *Item*** option. If you copy an existing item or create a new item, the IVI Configuration utility launches the corresponding wizard when you click the **OK** button. After the wizard executes, you return to the original Properties dialog box.

To view or modify the properties of the current child item, click the **Properties** button. The Properties dialog box for the current child item appears. After you configure the properties, click the **OK** button to return to the original Properties dialog box.

The remainder of this section describes the information you configure in the Properties dialog box for each IVI configuration item.

Logical Name Properties

You configure the properties of a logical name with the Properties dialog box shown in Figure 4-4.



Logical Name Properties - SampleDMM

SampleDMM

Description: Sample DMM Logical Name

Class: IviDmm

Virtual Instrument: FL45

Figure 4-4. Logical Name Properties Dialog Box

The Logical Name Properties dialog box displays the logical name, a description, the instrument class, and the virtual instrument that the logical name uses. The dialog box allows you to edit the description of the logical name, edit the properties of the virtual instrument, and select a different virtual instrument for the logical name to use.

You cannot modify the instrument class for the logical name. The class of the virtual instrument to which the logical name refers determines the instrument class for the logical name.

To view or modify the properties of the virtual instrument to which the logical name currently refers, click the **Properties** button. The Properties dialog box for the virtual instrument appears. Refer to the *Virtual*

Instrument Properties section of this chapter for more information regarding the Virtual Instrument Properties dialog box.

To configure the logical name to use a different virtual instrument, click the **Change** button. The Change Virtual Instrument dialog box appears. Refer to the *Changing Items that Another Item References* section in this chapter for more information regarding the Change Virtual Instrument dialog box.

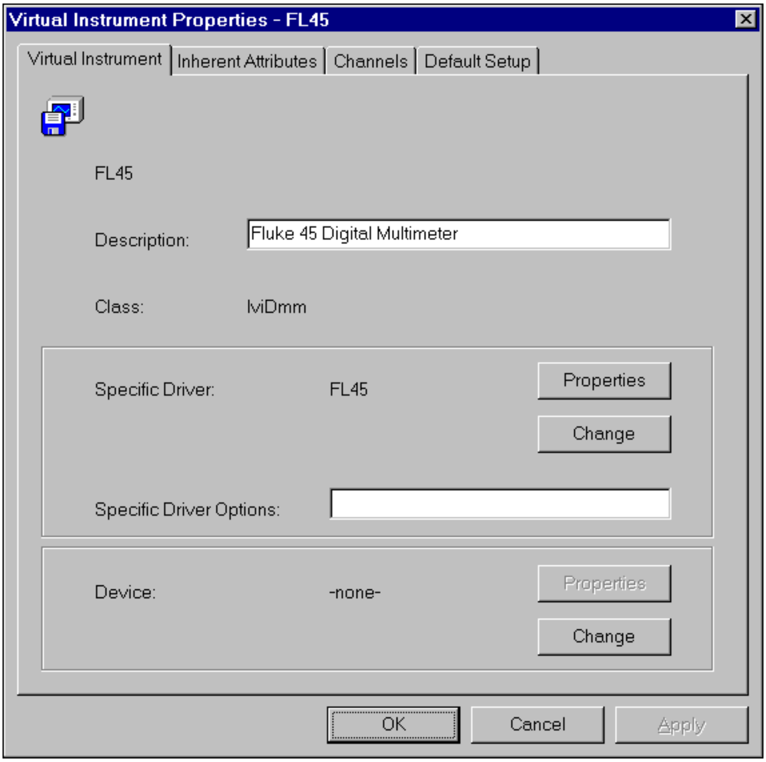
To change an actual logical name, such as changing `DMM1` to `MyMultimeter`, first select the item in the Explorer view. Then click once on the selected item to begin editing the name.

Virtual Instrument Properties

The Properties dialog box for a virtual instrument has tabs that allow you to configure the various components of the virtual instrument.

Virtual Instrument Tab

To configure the device and specific driver for the virtual instrument, click the Virtual Instrument tab shown in Figure 4-5.



Virtual Instrument Properties - FL45

FL45

Description: Fluke 45 Digital Multimeter

Class: IviDmm

Specific Driver: FL45

Specific Driver Options:

Device: -none-

Figure 4-5. Virtual Instrument Tab

The Virtual Instrument tab of the dialog box displays the virtual instrument name, class, description, device, and specific driver. Use this dialog box to edit the description and to configure the device and specific driver.

You cannot modify the instrument class for the virtual instrument. The class of the driver to which a virtual instrument refers determines the class of the virtual instrument.

To view or modify the properties of the device to which the virtual instrument currently refers, click the **Properties** button. The Properties dialog box for the device appears. Refer to the *Device Properties* section in this chapter for more information regarding the Device Properties dialog box.

To configure the virtual instrument to use a different device, click the **Change** button. The Change Device dialog box appears. Refer to the *Changing Items that Another Item References* section in this chapter for more information regarding the Change Device dialog box.

To edit the properties of the driver that the virtual instrument is currently using, click the **Properties** button. The Properties dialog box for the driver appears. Refer to the *Specific Driver Properties* section in this chapter for more information regarding the Specific Driver Properties dialog box.

To configure the virtual instrument to use a different driver, click the **Change** button. The Change Driver dialog box appears. Refer to the *Changing Items that Another Item References* section in this chapter for more information regarding the Change Driver dialog box.

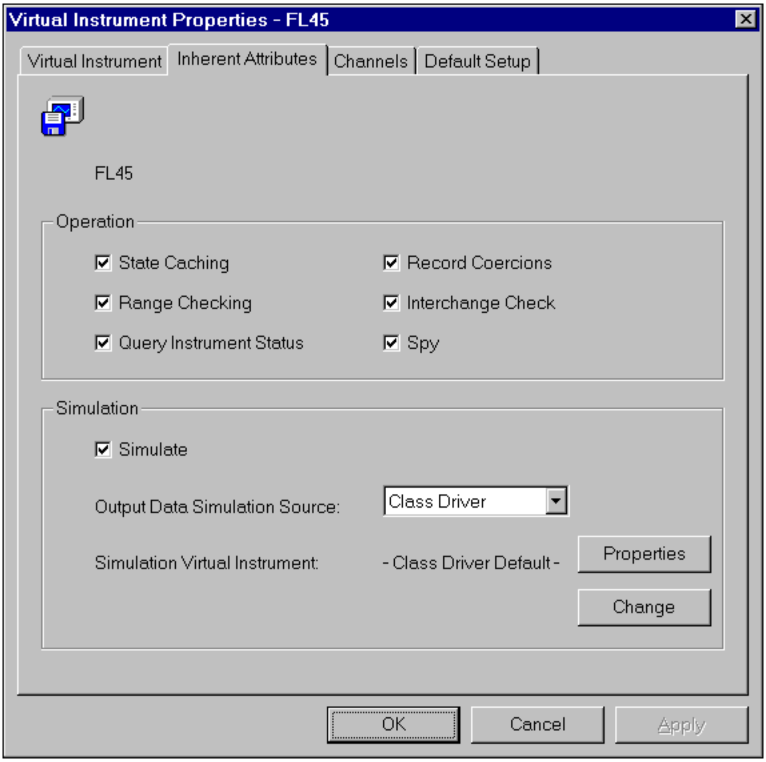
Sometimes you must specify particular driver options at initialization time. If you are using simulation, for example, you might want to specify a particular instrument model from among a family of instruments the driver supports. You can use the **Driver Options** control to specify a driver-specific setup string. Refer to the documentation for the specific driver for more information regarding the format of the driver-specific setup string.

Inherent Attributes Tab

To configure the inherent attributes of the virtual instrument, click the Inherent Attributes tab.

The Inherent Attributes tab appears in Figure 4-6.

Note Attributes that you set with the option string in either the `InitWithOptions` function or the Initialize With Options VI overwrite the settings in your Virtual Instrument Configuration. National Instruments recommends that you use the `init` function or the Initialize VI in conjunction with the Virtual Instrument Configuration settings.



FL45

Operation

☑ State Caching
☑ Range Checking
☑ Query Instrument Status
☑ Record Coercions
☑ Interchange Check
☑ Spy

Simulation

☑ Simulate

Output Data Simulation Source: Class Driver

Simulation Virtual Instrument: - Class Driver Default -

Figure 4-6. Inherent Attribute Properties Tab

Use the **Operation** controls group to configure most of the inherent IVI attributes, such as state caching and range checking. When you call the initialize function for a class driver, the class driver sets the inherent attributes to the values you specify.

Use the **Simulation** controls group to configure how the class driver simulates the specific instrument. To enable simulation, enable the Simulate checkbox.

The **Output Data Simulation Source** ring control specifies whether the class driver or the specific driver generates the simulation data for output parameters. If you select `Class Driver` from the **Output Data Simulation Source** ring control, the class driver controls the simulation data for output parameters. To do so, the class driver initializes a simulation driver that generates the simulation data. The class driver determines which simulation driver to initialize based on the contents of the Simulation Virtual Instrument indicator shown in Figure 4-6.

To view or modify the properties of the simulation virtual instrument to which the virtual instrument currently refers, click the **Properties** button. The Properties dialog box for the simulation virtual instrument appears. Refer to the *Simulation Virtual Instrument Properties* section in this chapter for more information regarding the Simulation Virtual Instrument Properties dialog box.

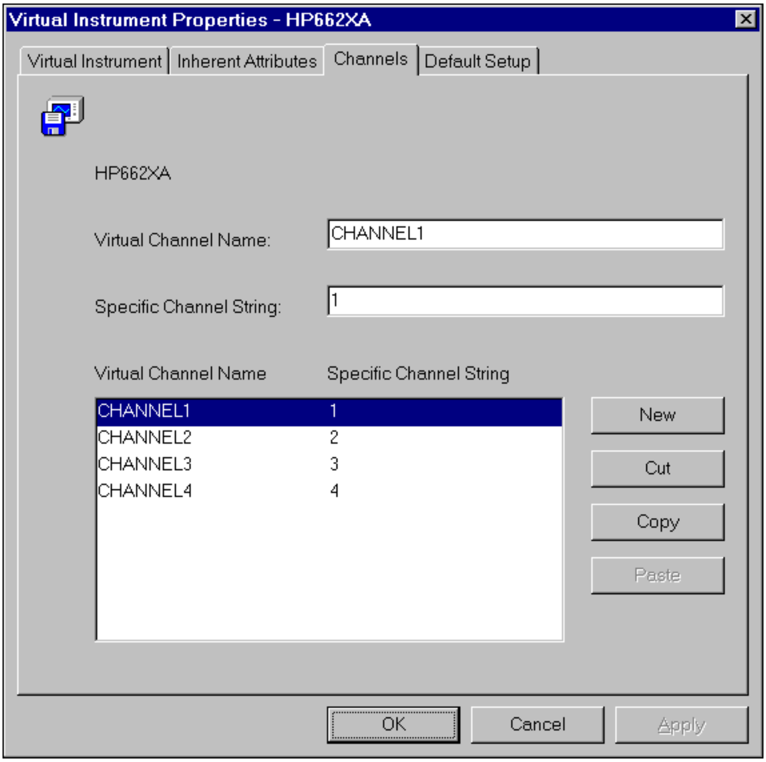
To configure the virtual instrument to use a different simulation virtual instrument, click the **Change** button. The Change Simulation Virtual Instrument dialog box appears. Refer to the *Changing Items that Another Item References* section in this chapter for more information regarding the Change Simulation Virtual Instrument dialog box.

If you select `Specific Driver` from the **Output Data Simulation Source** ring control, the specific driver generates simulation data for output parameters.

Channels Tab

If the class of the virtual instrument allows for multiple channels, you must configure virtual channel names. You associate the particular channel strings that the specific driver uses with the virtual channel names that your application program uses. If you change the instrument to which the virtual instrument refers and the driver for the new instrument uses different channel strings than the old instrument, you must reconfigure the virtual channel names.

To configure the virtual channel names of the virtual instrument, click the Channels tab shown in Figure 4-7.



Virtual Instrument Properties - HP662XA

HP662XA

Virtual Channel Name: CHANNEL1

Specific Channel String: 1

Virtual Channel Name	Specific Channel String
CHANNEL1	1
CHANNEL2	2
CHANNEL3	3
CHANNEL4	4

Figure 4-7. Virtual Channel Properties Tab

To create a new virtual channel name perform the following steps:

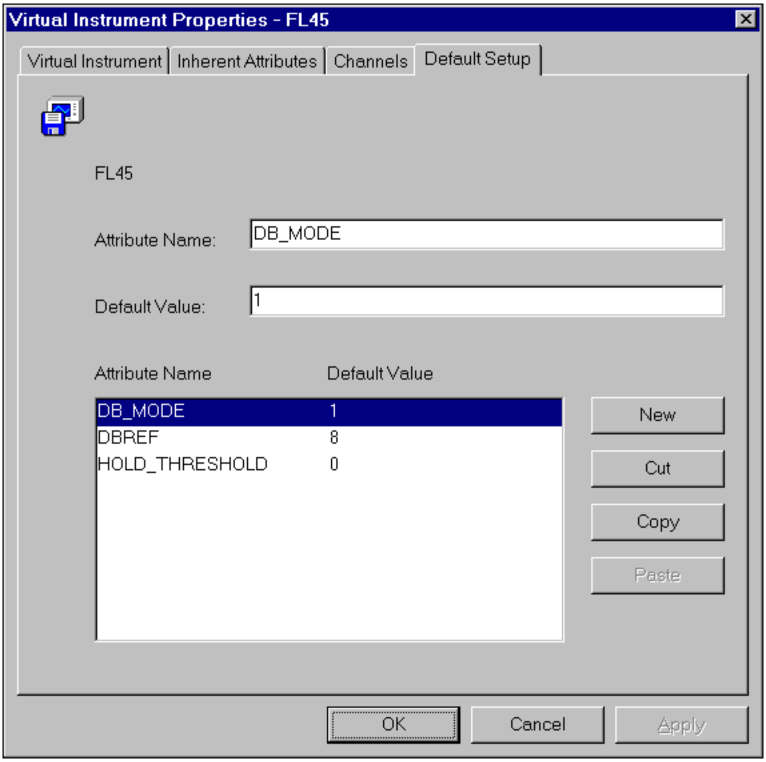
1. Click the **New** button.
2. Enter the new virtual channel name in the **Virtual Channel Name** control.
3. Enter the instrument-specific channel string that you want to associate with the virtual channel name in the **Specific Channel String** control. Refer to the documentation for the specific driver that you configure in the Virtual Instrument tab for information regarding the specific channel strings for the driver.

To modify an existing virtual channel name entry, select the particular entry in the list box. The contents of the entry appear in the **Virtual Channel Name** and **Specific Channel String** controls.

You can cut, copy, and paste entries in the list box by using the **Cut**, **Copy**, and **Paste** buttons that are to the right of the list box.

Default Setup Tab

To configure the default setup of the virtual instrument, click the Default Setup tab shown in Figure 4-8.



FL45

Attribute Name: DB_MODE

Default Value: 1

Attribute Name	Default Value
DB_MODE	1
DBREF	8
HOLD_THRESHOLD	0

Figure 4-8. Default Attribute Setting Properties Tab

The Default Setup tab allows you to configure attribute settings that are external to your program. When you call the initialize function for a class driver, the class driver invokes the initialize function in the specific driver. After the initialize function in the specific driver executes, the class driver applies the default setup that you specify in the Default Setup tab of this dialog box. The default setup feature is very useful for setting the initial values of instrument-specific attributes.

To specify the default setting for a particular attribute perform the following steps:

1. Click the **New** button.
2. Enter the name of the attribute in the **Attribute Name** control. You do not have to specify the prefix for the attribute. For example, you can

enter _ATTR_DB_MODE or DB_MODE instead of FL45_ATTR_DB_MODE. Refer to the documentation for the specific driver for a complete list of attributes that you can configure.

Note LabVIEW users must use the LabWindows/CVI attribute notation—not LabVIEW property naming conventions—when they make entries in the Default Setup tab.

3. Enter the initial value for the attribute in the **Default Value** control. Refer to the documentation for the specific driver for a complete description of the valid values for the attribute.

Note The default value that you specify for an attribute must be the actual value. You cannot use the name of a macro to specify a value. For ViBoolean attributes, enter True to enable the attribute, and enter False to disable the attribute.

Specify default values for multichannel attributes with a list of *channel*:*value* pairs that are delimited with semicolons. To apply a value to all channels that you do not configure explicitly, insert an asterisk (*) instead of a channel name.

Consider the following default value setting for a specific driver that supports four channels, ch1, ch2, ch3, and ch4:

```
ch1:True;ch2:True;*:False
```

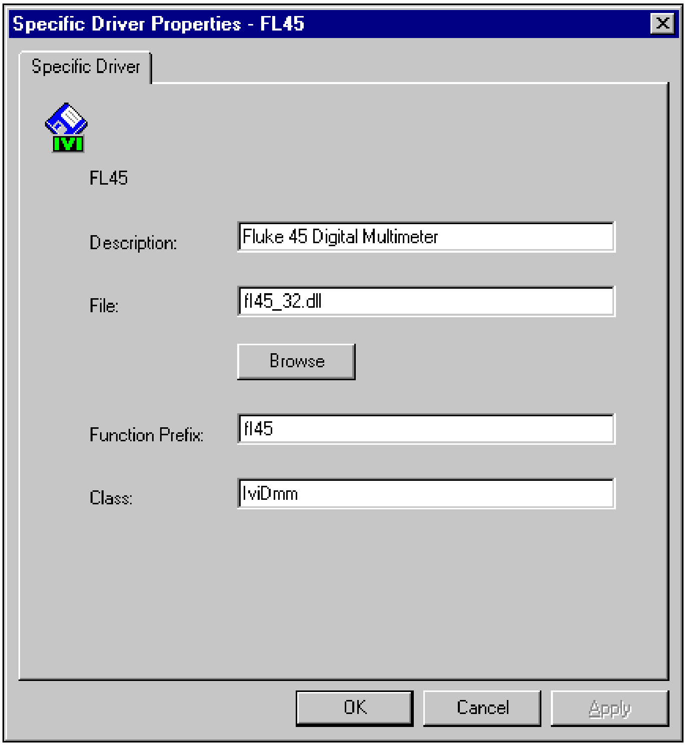
The preceding string specifies the following values for each channel: ch1 = True, ch2 = True, ch3 = False and ch4 = False.

In some cases, a specific driver might create additional channels after you call the initialize function. When you use an asterisk (*) to specify a default value for a multichannel attribute, the class driver applies this value to each new channel that the specific driver creates.

The class driver applies the default values in the order in which they appear in the list box. It is best to place a dependent attribute lower in the list than the attributes upon which it depends. For example, a FREQUENCY attribute might have a maximum value of 1000 when you set the MODE attribute to 1 and a maximum value of 2000 when you set the MODE attribute to 2. If you want to set the initial value of the MODE attribute to 2 and the initial value of the FREQUENCY attribute to 2000, the MODE attribute must appear above the FREQUENCY attribute in the list box.

Specific Driver Properties

You configure the properties of a specific driver with the Properties dialog box shown in Figure 4-9.



Description: Fluke 45 Digital Multimeter

File: fl45_32.dll

Function Prefix: fl45

Class: IviDmm

Figure 4-9. Specific Driver Properties Dialog Box

You can configure the description, driver location, class, and function prefix in the Properties dialog box for a specific driver.

The class is the name of the class driver with which the specific driver is compliant. If, for example, the driver is compliant with the IviDmm class driver, enter `IviDmm` in the **Class** control. If the driver is compliant with more than one class, enter the name of each class separated by commas, for example `IviFgen`, `IviScope`.

The class that you specify determines the class of each virtual instrument that refers to the particular specific driver. The class of each virtual instrument, in turn, determines the class of the logical names that refer to the virtual instrument. You can leave the **Class** control blank, but the class information is helpful when you swap instruments.

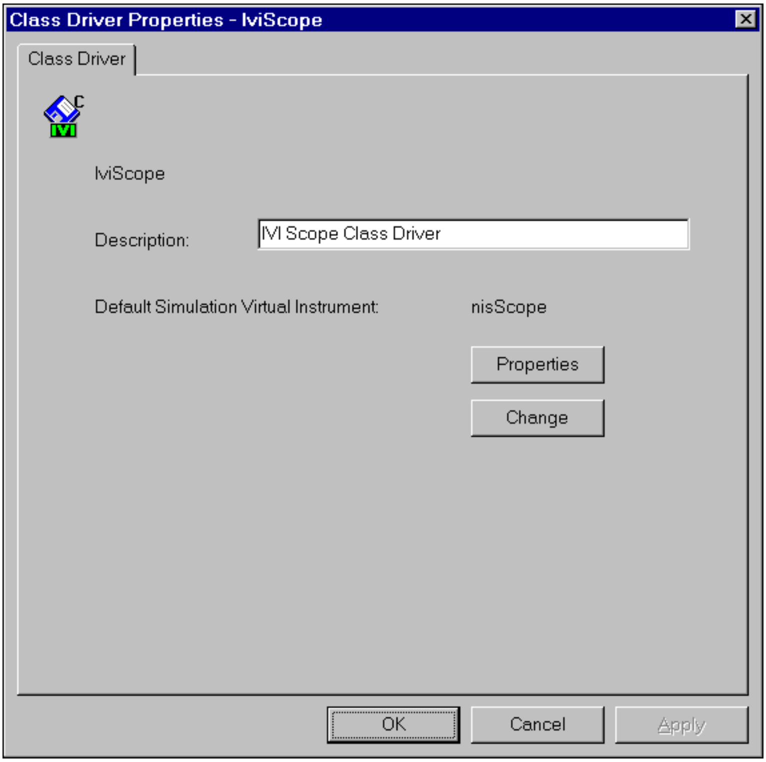
When you call an initialize function for a class driver, the class driver loads the specific driver module that you specify in the **File** control. You must enter the complete filename and path for the specific driver. You can use the **Browse** button to select a particular specific driver file. The file must be a `.dll`, `.lib`, or `.obj` file. If you are using a `.dll` file that is in the system search path, you do not have to specify a path. Instead, you can specify only the name of the `.dll` file. To use a `.lib` or `.obj` file, you must have the LabWindows/CVI run-time engine on your system. Furthermore, if you use a `.lib` or `.obj` file, you must enter the complete path and filename for the driver.

If you are running your program in the LabWindows/CVI environment, you can also use a `.c` file. If you use a `.c` file, you must include the file in the LabWindows/CVI project. You have more debugging options when you use a `.c` file in LabWindows/CVI.

After the class driver loads the specific driver, the class driver locates functions in the specific driver module by name. The class driver creates the function names on which to search by appending the names of class-defined functions to the function prefix that you specify in the **Function Prefix** control. The function prefix is case sensitive.

Class Driver Properties

You configure the properties of a class driver with the Properties dialog box shown in Figure 4-10.



Class Driver Properties - IviScope

IviScope

Description: IVI Scope Class Driver

Default Simulation Virtual Instrument: nisScope

Figure 4-10. Class Driver Properties Dialog Box

Use the Properties dialog box for a class driver to configure the description and the default simulation virtual instrument for the class driver.

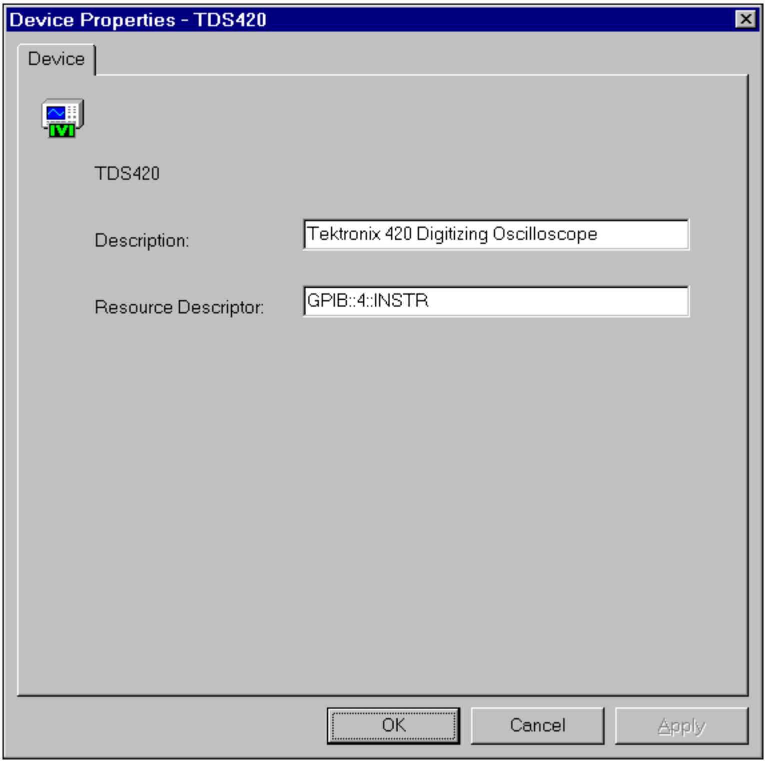
If you set the **Simulation Virtual Instrument** control on the Inherent Attribute tab for the virtual instrument that you are simulating to `Class Driver Default`, the class driver uses the default simulation virtual instrument to generate simulated output data.

To edit the properties of the simulation virtual instrument to which the class driver currently refers, click the **Properties** button. The Properties dialog box for the simulation virtual instrument appears. Refer to the *Simulation Virtual Instrument Properties* section in this chapter for more information regarding the Simulation Virtual Instrument Properties dialog box.

To configure the class driver to use a different simulation virtual instrument, click the **Change** button. The Change Simulation Virtual Instrument dialog box appears. Refer to the *Changing Items that Another Item References* section in this chapter for more information regarding the Change Simulation Virtual Instrument dialog box.

Device Properties

Configure the properties of a device with the Properties dialog box shown in Figure 4-11.



Description: Tektronix 420 Digitizing Oscilloscope

Resource Descriptor: GPIB::4::INSTR

Figure 4-11. Device Properties Dialog Box

Use the Properties dialog box to configure the description and resource descriptor for a particular device.

The resource descriptor specifies the interface and address of the device. Refer to the following table for the exact grammar to use. Optional fields are shown in square brackets ([]).

Table 4-2. Valid Resource Descriptors

Interface	Address Format
GPIB	`GPIB[board]::<primary address>[::secondary address][::INSTR]`
VXI	`VXI[board]::<logical address>[::INSTR]`
GPIB-VXI	`GPIB-VXI[board]::<logical address>[::INSTR]`
PXI	`PXI[bus]::deviceNumber[::functionNumber][::INSTR]`
Serial	`ASRL<port>[::INSTR]`
DAQ	`DAQ::<device number>[::INSTR]`

Use the `GPIB` keyword for GPIB instruments. Use the `VXI` keyword for VXI instruments that you control through embedded or MXIbus controllers. Use the `GPIB-VXI` keyword for VXI instruments that you control through a GPIB-VXI board. Use the `ASRL` keyword for serial instruments. Use the `DAQ` keyword for NI-DAQ devices and `PXI` for PXI devices.

If you do not specify a value for an optional field, the IVI engine uses the following values.

Table 4-3. Default Values for Fields

Optional String Segments	Default Value
board	`0`
secondary address	`none`
INSTR	`INSTR`

The following table contains examples of valid resource names.

Table 4-4. Valid Resource Names

Resource Name	Description
`GPIB::22::INSTR`	GPIB board 0, primary address 22 no secondary address
`GPIB::22::5::INSTR`	GPIB board 0, primary address 22 secondary address 5
`GPIB1::22::5::INSTR`	GPIB board 1, primary address 22 secondary address 5
`VXI::64::INSTR`	VXI board 0, logical address 64

Table 4-4. Valid Resource Names (Continued)

Resource Name	Description
`VXI1::64::INSTR`	VXI board 1, logical address 64
`GPIB-VXI::64::INSTR`	GPIB-VXI board 0, logical address 64
`GPIB-VXI1::64::INSTR`	GPIB-VXI board 1, logical address 64
`ASRL2::INSTR`	COM port 2
`DAQ::1::INSTR`	DAQ device 1
`PXI::18::INSTR`	PXI device number 18

Simulation Virtual Instrument Properties

Configure the properties of a simulation virtual instrument in the same way that you configure the properties of a virtual instrument. The configuration of a virtual instrument and the configuration of a simulation virtual instrument differ as follows:

- Simulation virtual instruments reference simulation drivers instead of specific drivers.
- Simulation virtual instruments do not reference a device.
- You cannot configure the IVI inherent attributes of a Simulation Virtual Instrument.
- You cannot configure the virtual channel names for a simulation virtual instrument. The simulation virtual instrument uses the virtual channel names for the virtual instrument it simulates.

Refer to the *Virtual Instrument Properties* section in this chapter for more information regarding the configuration of virtual instruments.

Simulation Driver Properties

You configure the properties of a simulation driver in the same way that you configure the properties of a specific driver. Refer to the *Specific Driver Properties* section of this chapter for more information regarding the configuration of drivers.

Creating a Logical Name

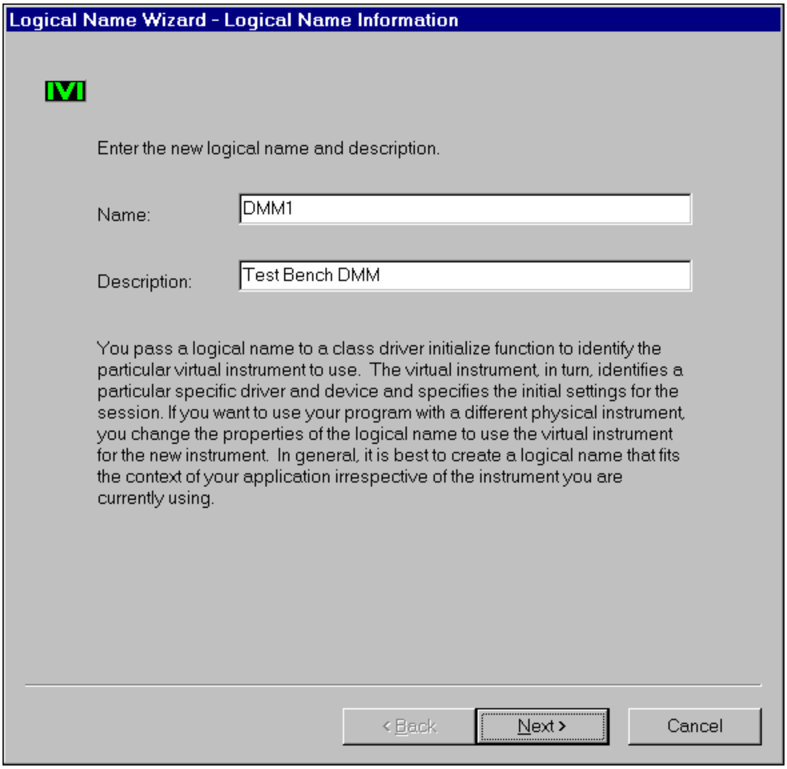
Use the Add Logical Name wizard to create a new logical name. Through a series of panels, the wizard prompts you for all the information necessary to create a new logical name and create or configure all the IVI configuration items that the logical name refers to.

Step 1: Launch the Logical Name Wizard

To invoke the wizard, launch the IVI Configuration utility, open the `IVI` folder, and perform one of the following actions:

- Double-click the Add Logical Name icon.
- Right-click the `Logical Names` folder and select **Insert** from the context menu.

The Logical Name Wizard panel is shown in Figure 4-12.



Enter the new logical name and description.

Name: DMM1

Description: Test Bench DMM

You pass a logical name to a class driver initialize function to identify the particular virtual instrument to use. The virtual instrument, in turn, identifies a particular specific driver and device and specifies the initial settings for the session. If you want to use your program with a different physical instrument, you change the properties of the logical name to use the virtual instrument for the new instrument. In general, it is best to create a logical name that fits the context of your application irrespective of the instrument you are currently using.

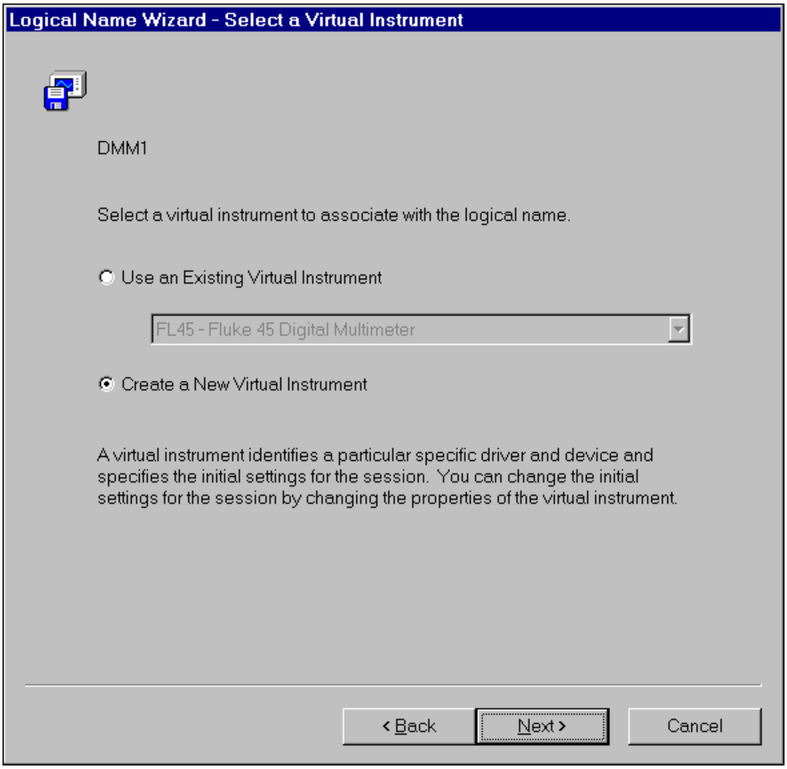
Figure 4-12. Logical Name Wizard

Enter the name and description you want to use for the new logical name. After the wizard executes, the new logical name appears in the `Logical Name` folder with the name you enter in the **Name** control.

Click the **Next** button to continue. At any time, you can click the **Back** button to return to a previous panel and change the information.

Step 2: Select a Virtual Instrument

The Select a Virtual Instrument panel is shown in Figure 4-13.



Logical Name Wizard - Select a Virtual Instrument

DMM1

Select a virtual instrument to associate with the logical name.

○ Use an Existing Virtual Instrument

FL45 - Fluke 45 Digital Multimeter

● Create a New Virtual Instrument

A virtual instrument identifies a particular specific driver and device and specifies the initial settings for the session. You can change the initial settings for the session by changing the properties of the virtual instrument.

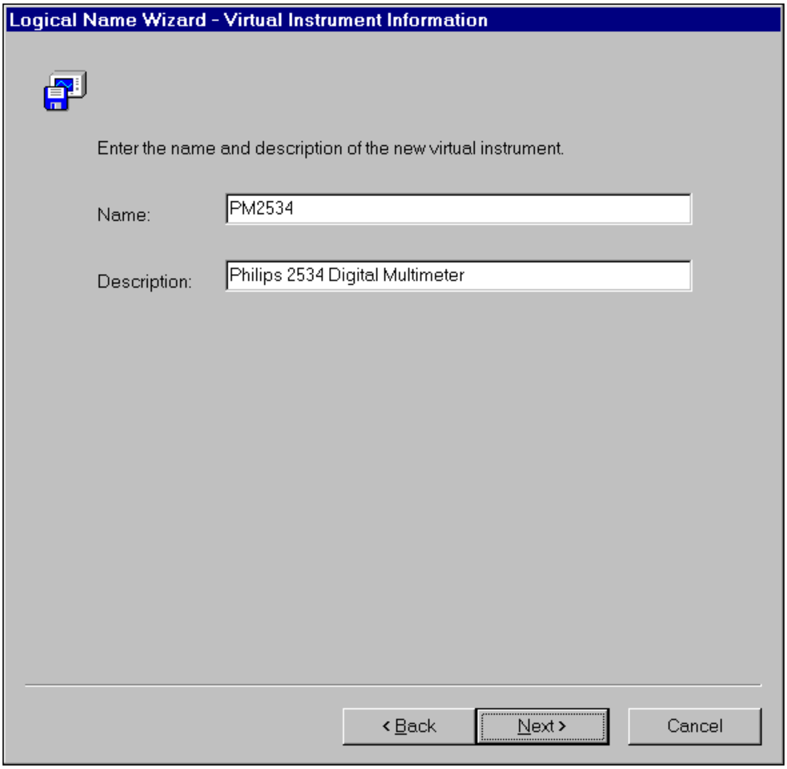
Figure 4-13. Select a Virtual Instrument Panel

The Select a Virtual Instrument panel prompts you to select the virtual instrument you want to use for the new logical name. You can select an existing virtual instrument or create a new virtual instrument.

To use an existing virtual instrument, select the **Use an Existing Virtual Instrument** option and choose a virtual instrument from the ring control below the option. Then click the **Next** button. If the existing virtual instrument currently does not refer to a device, the wizard jumps to Step 4. Otherwise, the wizard jumps to Step 5.

To create a new virtual instrument, select the **Create a New Virtual Instrument** option.

Click the **Next** button to continue. The Virtual Instrument Information panel appears in Figure 4-14.



Enter the name and description of the new virtual instrument.

Name: PM2534

Description: Philips 2534 Digital Multimeter

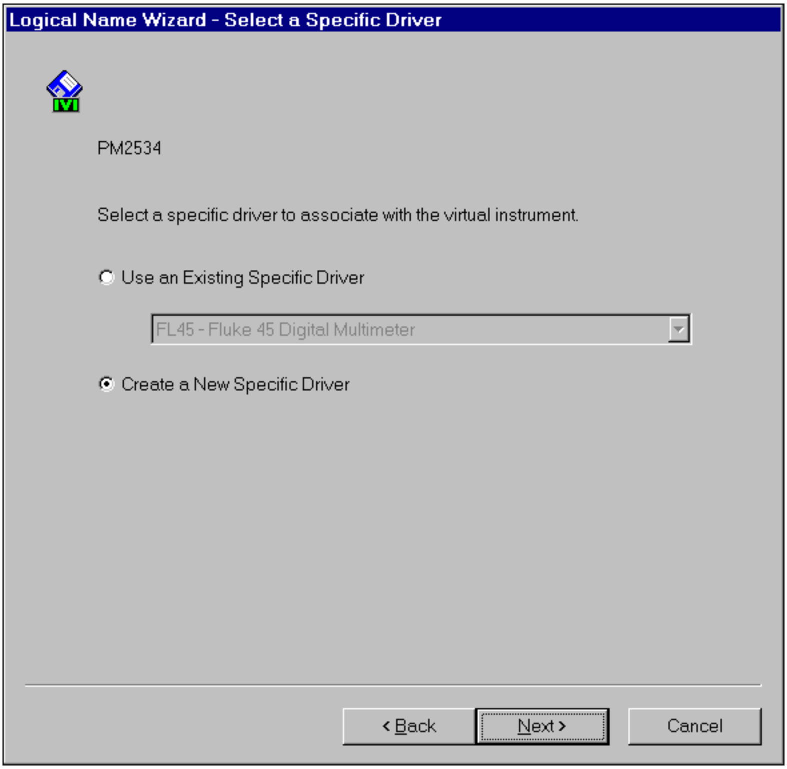
Figure 4-14. Virtual Instrument Information Panel

Enter a name that uniquely identifies the new virtual instrument, and enter a description. After the wizard executes, the new virtual instrument appears in the `Virtual Instruments` folder with the name that you specify in the **Name** control. The logical name that you create refers to the virtual instrument by that name.

Click the **Next** button.

Step 3: Select a Specific Driver

The Select a Specific Driver panel is shown in Figure 4-15.



Logical Name Wizard - Select a Specific Driver

PM2534

Select a specific driver to associate with the virtual instrument.

○ Use an Existing Specific Driver

FL45 - Fluke 45 Digital Multimeter

● Create a New Specific Driver

Figure 4-15. Select a Driver Panel

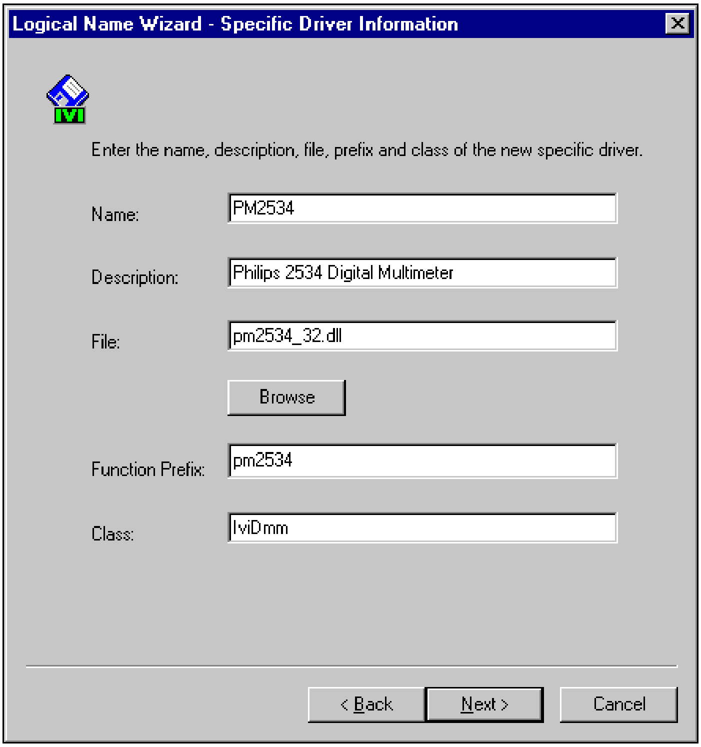
The Select a Specific Driver panel prompts you to select a specific driver. The class driver uses the driver you select to communicate with the device. The Select a Specific Driver panel operates similarly to the Select a Virtual Instrument panel. You can select an existing driver or create a new driver.

To use an existing driver, select the **Use an Existing Driver** option and select a driver from the ring control below the option. Then click the **Next** button. The wizard jumps to Step 4.

To create a new driver, select the **Create a New Driver** option.

Click the **Next** button to continue.

The Specific Driver Information panel appears in Figure 4-16.



Logical Name Wizard - Specific Driver Information

Enter the name, description, file, prefix and class of the new specific driver.

Name: PM2534

Description: Philips 2534 Digital Multimeter

File: pm2534_32.dll

Function Prefix: pm2534

Class: IviDmm

Figure 4-16. Specific Driver Information Panel

Enter the following information in the Specific Driver Information Panel:

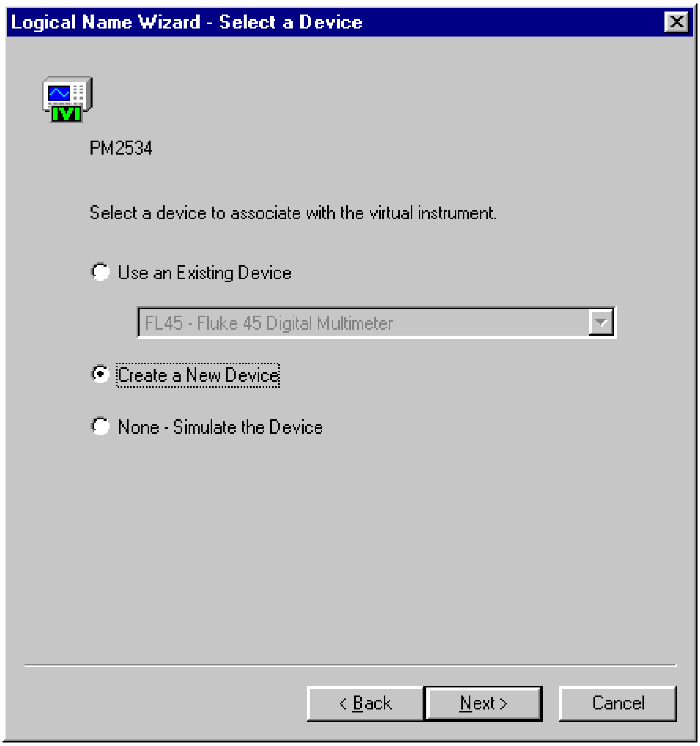
- A name that uniquely identifies the new driver
- A description for the driver
- The location and name of the driver file you want to load
- The class with which the driver is compliant
- The case-sensitive function prefix for the driver

After the wizard executes, the new specific driver appears in the `Instrument Drivers` folder with the name that you specify in the **Name** control. The virtual instrument refers to the specific driver by that name.

Click the **Next** button to continue.

Step 4: Select a Device

The Select a Device panel appears in Figure 4-17.



PM2534

Select a device to associate with the virtual instrument.

○ Use an Existing Device

FL45 - Fluke 45 Digital Multimeter

● Create a New Device

○ None - Simulate the Device

Figure 4-17. Select a Device Panel

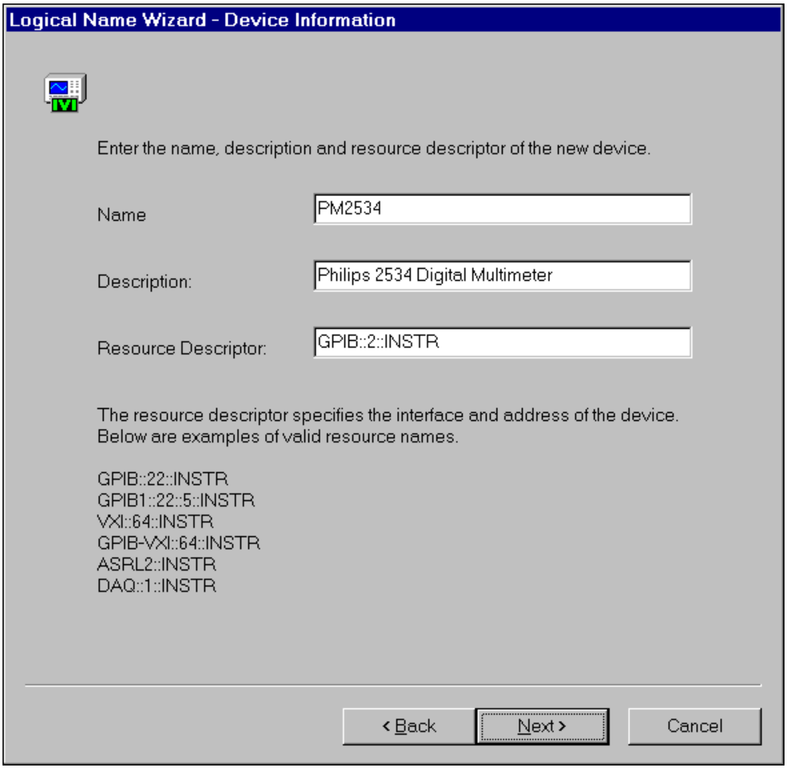
The Select a Device panel prompts you to select a device. The panel operates similarly to the Select a Virtual Instrument panel. You can select an existing device, create a new device, or select to simulate the device.

To use an existing device, select the **Use an Existing Device** option and choose an existing virtual instrument from the ring control.

To create a new device, select the **Create a New Device** option. To simulate the device, select the **None-Simulate the Device** option.

Click the **Next** button to continue.

The Device Information panel appears in Figure 4-18.



Logical Name Wizard - Device Information

Enter the name, description and resource descriptor of the new device.

Name: PM2534

Description: Philips 2534 Digital Multimeter

Resource Descriptor: GPIB::2::INSTR

The resource descriptor specifies the interface and address of the device.
Below are examples of valid resource names.

GPIB::22::INSTR
GPIB1::22::5::INSTR
VXI::64::INSTR
GPIB-VXI::64::INSTR
ASRL2::INSTR
DAQ::1::INSTR

Figure 4-18. Device Information Panel

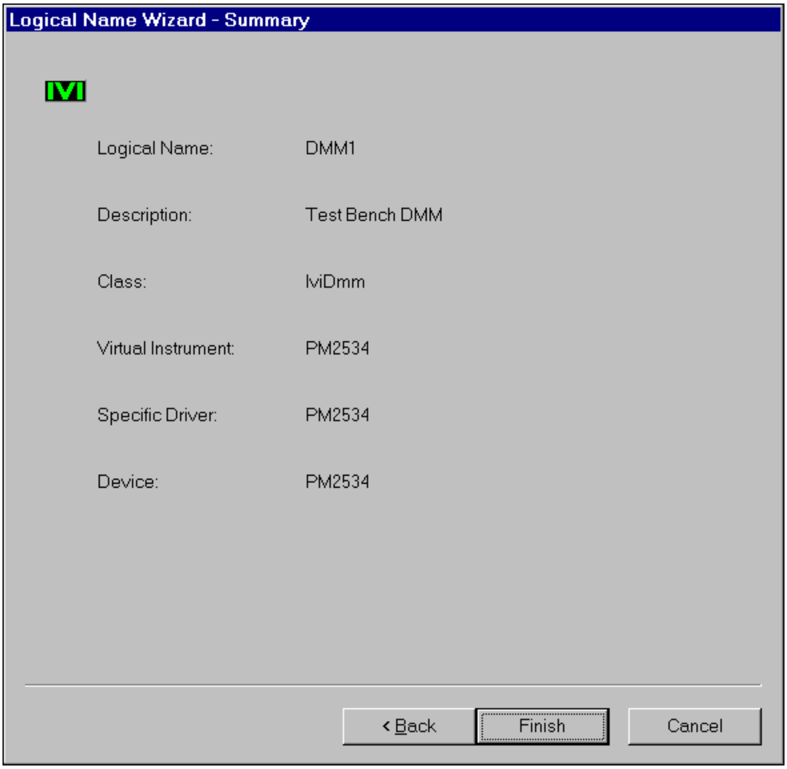
Enter a name that uniquely identifies the new device, a description, and the resource descriptor for the device. The resource descriptor is a string that identifies the address of the device in the system. For a GPIB device, the resource descriptor is a VISA resource descriptor such as `GPIB::2::INSTR`. Refer to the *Device Properties* section in this chapter for more information on the format of resource descriptors.

After the wizard completes, the new device appears in the `Devices` folder with the name that you specify in the **Name** control. The new virtual instrument refers to the device by that name.

Click the **Next** button to continue.

Step 5: Summary

The Summary panel appears in Figure 4-19.



Logical Name Wizard - Summary

Logical Name: DMM1

Description: Test Bench DMM

Class: IviDmm

Virtual Instrument: PM2534

Specific Driver: PM2534

Device: PM2534

Figure 4-19. Summary Panel

The Summary panel displays a summary of the new logical name. Click the **Finish** button to commit the changes and exit the wizard.

Swapping an Instrument

To swap an instrument, open the Properties dialog box for the corresponding logical name. In the dialog box, use the **Change** button to create or select the virtual instrument for the new physical instrument.

Advanced Topics

This section describes advanced configuration topics such as using multiple configuration files and creating IVI configuration information at run time.

Multiple IVI.INI Files

The information that IVI Configuration utility and IVI class drivers access resides in a file named `ivi.ini`. By default, the IVI Configuration utility and the IVI class drivers search for the `ivi.ini` file within the directory structure for VXI*plug&play*, `<`*`vxipnpbase`*`>\niivi\`.

In some cases, you might want to maintain multiple `ivi.ini` files to define different system configurations. To edit configuration information that you store in an `ivi.ini` file that resides in a directory other than the `<`*`vxipnpbase`*`>\niivi\` directory, perform the following steps:

1. Launch the IVI Configuration utility.
2. Select the `Measurement & Automation` folder.
3. Select **File»Open Configuration**. The Open Configuration dialog box appears.
4. In the Open Configuration dialog box, select the `IVI Configuration File` entry. Click the **Browse** button to select the directory that contains the `ivi.ini` file that you want to edit. If you select a directory that does not contain an `ivi.ini` file, the IVI Configuration utility prompts you to create a new file.

With IVI you can save the current `ivi.ini` file to another location and you can create a new `ivi.ini` file.

If you are using LabWindows/CVI and want to use a different `ivi.ini` file in your program, call the `Ivi_SetIviIniDir` function to specify the pathname of the directory where you want class drivers to search for the `ivi.ini` file. Call `Ivi_SetIviIniDir` prior to calling any initialize function for a class drivers. For information on the `Ivi_SetIviIniDir` function, refer to this function name in the *LabWindows/CVI Online Help*.

Run-Time Configuration

In some cases, you might not want to use a configuration file to specify your IVI configuration. The IVI *library* contains functions you can use to create run-time configuration information like the information that the IVI Configuration utility normally provides. The library also contains a function that writes the run-time configuration information to a file. The following run-time configuration functions are available in LabWindows/CVI.

Table 4-5. Run-Time Configuration Functions in LabWindows/CVI

Task	Function Name
Create or destroy logical names	`Ivi_DefineLogicalName` `Ivi_UndefLogicalName`
Create or destroy virtual instruments	`Ivi_DefineVInstr` `Ivi_UndefVInstr`
Create or destroy instrument drivers	`Ivi_DefineDriver` `Ivi_UndefDriver`
Create or destroy class drivers	`Ivi_DefineClass` `Ivi_UndefClass`
Create or destroy devices	`Ivi_DefineHardware` `Ivi_UndefHardware`
Save the run-time configuration to a file	`Ivi_WriteRunTimeDefinesToFile`

For more information regarding run-time configuration functions, refer to the *LabWindows/CVI Online Help*.

5

Class Driver Operation

This chapter describes how you use the IVI class drivers to develop interchangeable applications. This chapter also discusses strategies you can use to maximize the interchangeability of your application and to verify whether a new instrument can replace a particular instrument in your application.

Using IVI Class Drivers

After you configure your system with the IVI Configuration utility, you develop an interchangeable application by making calls to the IVI class drivers. The class drivers isolate your program from the specific drivers that communicate with the instruments.

Using Class Drivers in LabWindows/CVI

Use IVI class drivers in the LabWindows/CVI environment in the same way that you use other LabWindows/CVI instrument drivers. You can load and unload class drivers manually using the Instrument menu. To load a class driver do the following:

1. Select **Instrument»Load**.
2. In the Load Instrument dialog box, select the function panel (`.fp`) file for the class driver you want to load.

The class driver function panel files are in the `cvi\instr\iviclass` directory.

Figure 5-1 shows the **Instrument** menu after you load the class drivers.

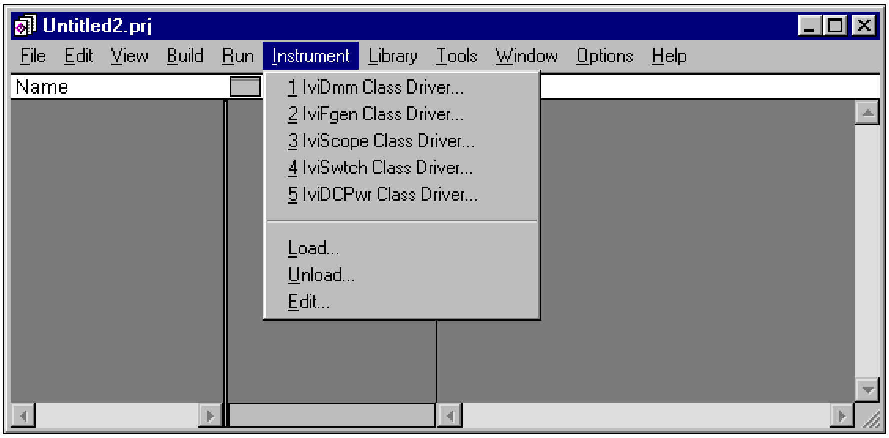

Untitled2.prj
File Edit View Build Run Instrument Library Tools Window Options Help
Name
1 IviDmm Class Driver...
2 IviFgen Class Driver...
3 IviScope Class Driver...
4 IviSwtch Class Driver...
5 IviDCPwr Class Driver...
Load...
Unload...
Edit...

Figure 5-1. LabWindows/CVI Instrument Menu

You do not have to include the class drivers that you load through the **Instrument** menu in your project, and you can load and unload them at any time.

You incorporate the IVI class drivers into your project by selecting **File»Add to Project** in a Function Panel window or in the **Edit»Add to Project** menu of the Project window. The `.fp` file represents the class driver in the project list. If the `.fp` file is in the project list, LabWindows/CVI automatically loads the class driver when you open the project and removes the class driver when you unload the project.

A class driver function panel contains a function panel window for each function that the class driver exports. With the function panel windows, you can interactively call class driver functions and automatically generate code for your application.

The class drivers have high-level and low-level functions. With the high-level functions you can easily initialize and close the instrument, configure the instrument, control instrument operations, and retrieve measurements. High-level configure functions set multiple instrument attributes in a single operation.

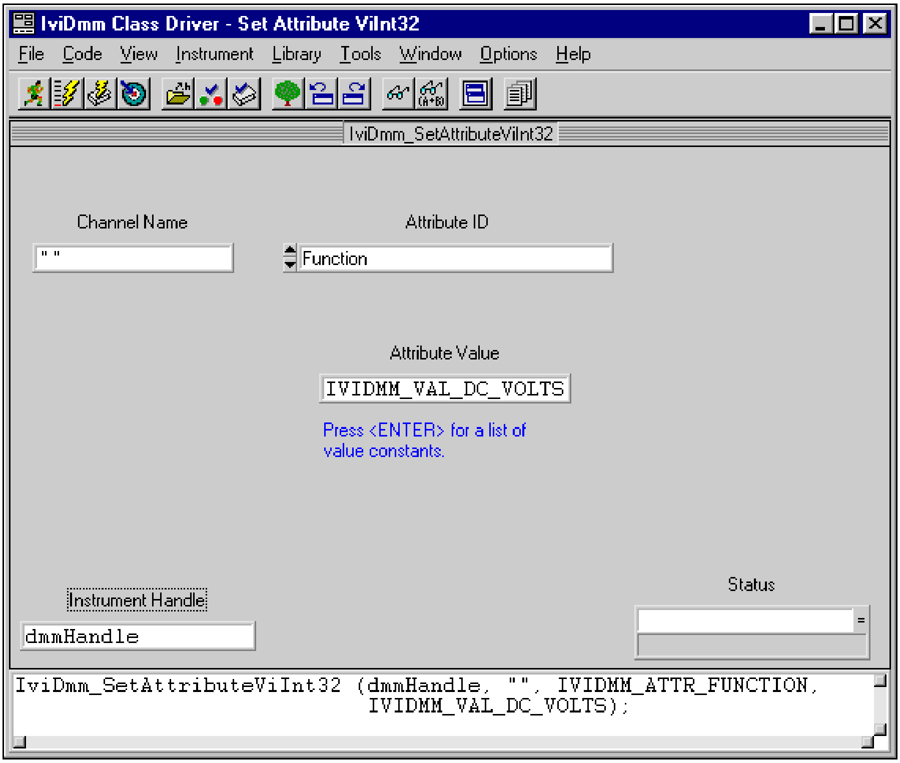
For example, to set up and take a measurement using a DMM, you might use the following high-level statements in your program:

```
ViReal64 reading;
ViSession dmmHandle;

IviDmm_Init ("DMM1", VI_TRUE, VI_TRUE, &dmmHandle);
IviDmm_ConfigureMeasurement (dmmHandle,
                    IVIDMM_VAL_DC_VOLTS_RATIO,
                    IVIDMM_VAL_AUTO_RANGE_ON, 0.0006));

IviDmm_ConfigureTrigger (dmmHandle,
                         IVIDMM_VAL_IMMEDIATE, 0.0));
IviDmm_Read (dmmHandle, 5000, &reading));
IviDmm_close (dmmHandle);
```

With the low-level functions of the class drivers, you can access individual instrument driver attributes. For example, Figure 5-2 shows the Set Attribute function panel for the IviDmm class driver.


IviDmm Class Driver - Set Attribute ViInt32
File Code View Instrument Library Tools Window Options Help
IviDmm_SetAttributeViInt32
Press <ENTER> for a list of value constants.
Status

Channel Name: " "

Attribute ID: Function

Attribute Value: IVIDMM_VAL_DC_VOLTS

Instrument Handle: dmmHandle

```
IviDmm_SetAttributeViInt32 (dmmHandle, "", IVIDMM_ATTR_FUNCTION,
                            IVIDMM_VAL_DC_VOLTS);
```

Figure 5-2. Set Attribute Function Panel Window

Refer to the *LabWindows/CVI User Manual* for complete information on how to use function panels in LabWindows/CVI.

Using Class Drivers in LabVIEW

Use IVI class drivers in the LabVIEW environment in the same way that you use other LabVIEW instrument drivers. The IVI class drivers are in the IVI Class Drivers palette. To access the IVI Class Drivers palette, select **Functions»InstrumentI/O»IVI Class Drivers**.

Figure 5-3 shows the IVI Class Drivers palette.

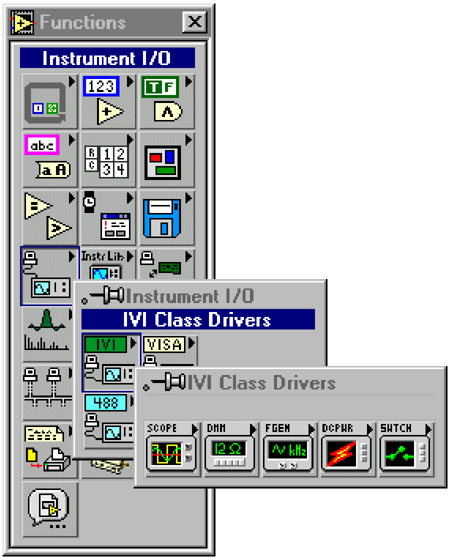


Figure 5-3. IVI Class Drivers Palette

Each IVI class driver has a subpalette. Each subpalette contains all the VIs for the corresponding class driver. The LabVIEW class drivers have VIs that perform the same operations as the LabWindows/CVI class drivers. To access a VI for a particular class driver, select the subpalette that corresponds to the class driver in the IVI Class Drivers palette.

Figure 5-4 shows a code example that uses the IviDmm class driver to configure a DMM and take a measurement.

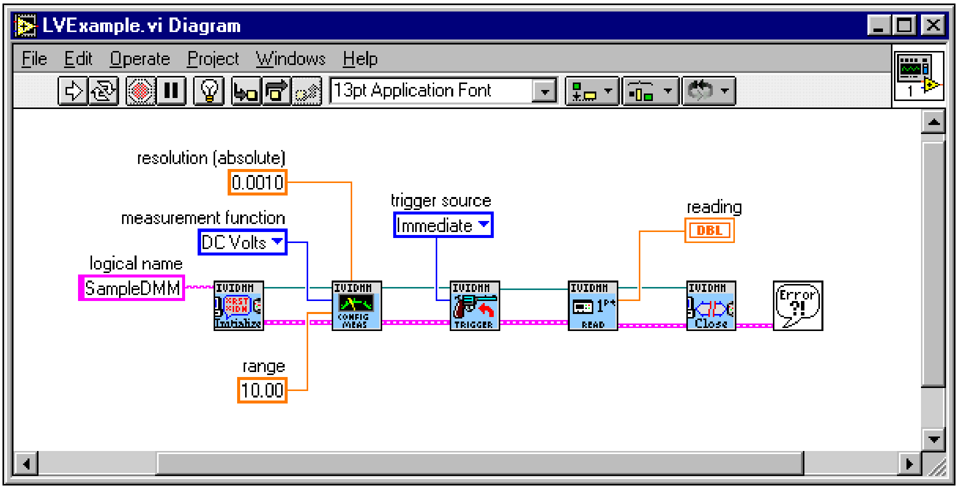


Figure 5-4. LabVIEW IviDmm Class Driver Example

In LabVIEW, you can access individual attributes with the property node.

Figure 5-5 shows how you can use the property node to access instrument driver attributes. After LabVIEW calls the IviDMM Initialize VI, it uses the Property Node to set values for the Function, Trigger Source, and Range.

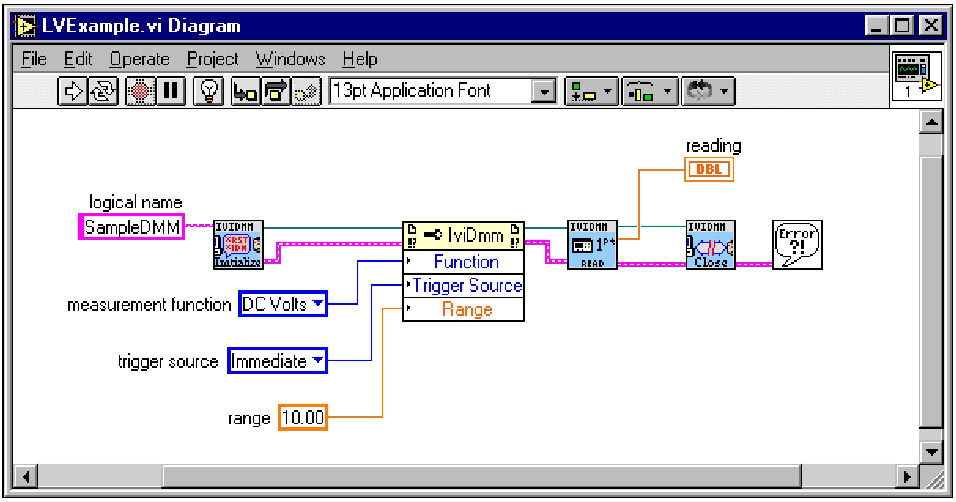


Figure 5-5. LabVIEW Property Node

Refer to the *VISA Library Reference* section of the *LabVIEW Function and VI Reference Manual* for complete information on how to use the property node.

Developing an Interchangeable Application

This section describes issues to consider when you use IVI class drivers to develop an instrument-independent test program. By following the guidelines in this section, you can maximize the potential for your application to work with other instruments.

Using Logical Names and Virtual Channel Names

You use logical names to identify a physical instrument and specific driver without including instrument-specific information in your test program. In general, it is best to create a logical name that fits the context of your application, no matter which specific instrument you use.

For example, the logical name `DMM` fits the context of a digital multimeter application in which you swap two or more digital multimeters, such as a Fluke 45 and an HP34401A. In contrast, the logical name `FL45` or `HP34401A` does not fit this context because these names are too specific.

When you create a logical name, you must consider whether you want to use it across multiple applications. One approach is to create a logical name that is global to all applications in your system. Another approach is to create a unique logical name for each application.

If you create global logical names, you can quickly reconfigure your entire system. When you replace an instrument, you have to reconfigure only one logical name in the IVI Configuration utility. On the other hand, any change to the configuration of the logical name in the IVI Configuration utility affects all applications in your system. Thus, your freedom to replace an instrument for any application is limited by the sum of the requirements of all the applications that use the logical name. Also, this approach requires that you coordinate the development of the applications that you run on your system so they all use the same set of logical names.

If you create a unique logical name for each application, you can customize the configuration of the instrument for each application. On the other hand, it may take you longer to reconfigure your system if you swap an instrument that many applications use. Furthermore, this approach requires that you coordinate the development of the applications that you run on your system so that they do not use the same logical names.

With virtual channel names you identify a particular channel of an instrument without using instrument-specific channel strings. Therefore, you must give the same considerations to your selection of virtual channel names as you do to your selection of logical names.

Using High-Level Configuration Functions Rather Than Setting Individual Attributes

IVI class drivers have both high-level and low-level configuration functions. It is best to use the high-level functions to configure an instrument. The high-level configuration functions group the setting of related attributes into a single operation. When you call a high-level configuration function in a class driver, the class driver invokes the corresponding function in the specific driver. The specific driver is responsible for setting the attributes in the correct order for the particular instrument. Also, the specific driver can handle complex interactions between multiple attributes for the instrument. If you swap instruments, the new specific driver sets the attributes in the correct order and handles attribute interactions for the new instrument.

With the low-level configuration functions you set values for individual attributes. When you use the low-level functions to manipulate attributes, you must understand the relationships and interactions between the attributes for a particular instrument. You might have to set the attribute values in a particular order for your program to work with a particular instrument. When you replace the instrument, the attribute order dependencies are likely to change. Therefore, you have to change the order in which your program sets the attributes when you swap the instrument.

Minimize the Use of Extension Capability Groups

It is best to use only the extension capability groups that your test program requires. Minimizing the number of extension capability groups you access for a particular IVI session maximizes the number of instruments that you potentially can use with your application.

The class drivers divide the capabilities of an instrument class into capability groups. The capability groups contain functions and attributes that you use to access the features of that capability group. Each class driver defines a base capability group. A specific driver must implement the base capabilities group to be compliant with the class. Therefore, you always can use the base capabilities group in your program.

The other capability groups are extension capability groups. Extension capability groups represent the less common capabilities of the instrument class. Specific drivers are not required to implement the extension capability groups. Not all specific drivers implement the same set of extension capability groups. Each time you use a new extension capability group in your program, you limit the number of instruments that you can use in your application.

Completely Specify the State of the Instrument

To maximize interchangeability, you must completely specify the state of the attributes that affect the behavior of the instrument. If you do not, the behavior of your program depends on instrument-specific settings that can result from any of the following conditions:

- The power-on settings of the device
- The state that the instrument configures for an attribute as a result of your program configuring other attributes
- The state that a previous program configured for the instrument

If you fail to specify the state of the instrument completely, you increase considerably the chance that your program will not behave the same way when you swap instruments or you run your programs in a different order.

In general, after you access a particular capability group, you must configure all attributes of that capability group that affect the behavior of the instrument. Because all specific drivers that are compliant with a class implement the base capability group, you must completely specify that state of the base capabilities. After you access a particular extension capability group, you must configure all attributes of that extension group. Normally, you configure the attributes through one or more high-level configuration functions.

It is important to note that not all attributes of a particular capability group affect the behavior of the instrument. In some cases, if one attribute is set to a particular value, a second attribute no longer affects the behavior of the instrument. In such cases, it is not mandatory that you specify the state of the second attribute. However, National Instruments recommends that you specify the state of all attributes within the capability groups that you use, for the sake of long-term stability and reuseability.

If you set the `IVIDMM_ATTR_SAMPLE_COUNT` to `1`, you do not need to specify the values of the `IVIDMM_ATTR_SAMPLE_TRIGGER` and `IVIDMM_ATTR_SAMPLE_INTERVAL` attributes because these attributes do not affect the behavior of the instrument.

Similarly, if you do *not* set the SAMPLE_TRIGGER to IVIDMM_VAL_INTERVAL, you do not need to specify the value of the IVIDMM_ATTR_SAMPLE_INTERVAL attribute.

Use the Development Mode Settings for Inherent Attributes

IVI class drivers have many features that help you debug and analyze your program. You enable these features by setting the *development mode* settings for inherent attributes.

After you complete the development of your application and verify that your program is working correctly, it is best to enable the *production mode* settings for the inherent attributes. The production mode settings maximize performance.

The following table lists the development mode and production mode settings for the inherent attributes.

Table 5-1. Development Mode and Production Mode Settings for Inherent Attributes

LabVIEW Property	Inherent Attribute	Development Mode Setting	Production Mode Setting
ClassPrefix Spy	*CLASSPREFIX*_ATTR_SPY	VI_TRUE	VI_FALSE
ClassPrefix Range Check	*CLASSPREFIX*_ATTR_RANGE_CHECK	VI_TRUE	VI_FALSE
ClassPrefix Query Instrument Status	*CLASSPREFIX*_ATTR_QUERY_INSTR_STATUS	VI_TRUE	VI_FALSE
ClassPrefix Interchange Check	*CLASSPREFIX*_ATTR_INTERCHANGE_CHECK	VI_TRUE	VI_FALSE
ClassPrefix Record Value Coercions	*CLASSPREFIX*_ATTR_RECORD_COERCIONS	VI_TRUE	VI_FALSE

Refer to Chapter 4, *Configuring Your System*, for information about how to use the IVI Configuration utility to set the development mode and production mode settings for inherent attributes.

- ***CLASSPREFIX*_ATTR_SPY**—Enables/disables spying. If you enable spying, the class driver logs all calls you make to class driver functions. You can view information about each call in the NI Spy utility. The *Analyzing Your Program with NI Spy* section in this chapter describes how to use the NI Spy utility to capture and view calls that your program makes to the class drivers.
- ***CLASSPREFIX*_ATTR_RANGE_CHECK**—Enables/disables range checking. If you enable range checking, the specific driver checks all

parameters that you pass to the class driver and specific driver functions.

- ***CLASSPREFIX*_ATTR_QUERY_INSTR_STATUS**—Enables/disables instrument status checking. If you enable instrument status checking, the specific driver checks the status of the instrument after each call your program makes to the class driver or specific driver.
- ***CLASSPREFIX*_ATTR_INTERCHANGE_CHECK**—Enables/disables interchangeability checking. Interchangeability checking verifies that your program produces the same behavior when you use it with a different instrument. If you enable interchangeability checking, the class driver queues warnings when it encounters instrument configurations that are not likely to produce the same behavior when you use a different instrument. Refer to Chapter 3, *IVI System Architecture*, for a complete description of interchangeability checking. Refer to the *IVI Class Driver Online Help* for a description of the interchangeability checking rules for each class driver.

  If you enable spying, you can use the NI Spy utility to view interchangeability warnings. The NI Spy utility highlights class driver functions that report interchangeability warnings in blue. If you disable spying, you can use the *CLASSPREFIX*_GetNextInterchangeWarning function to retrieve interchangeability warnings programmatically.
- ***CLASSPREFIX*_ATTR_RECORD_COERCIONS**—Enables/disables coercion recording. In many cases, specific drivers coerce the value you specify for a function parameter or attribute to a value the instrument can accept. Specific drivers may coerce a value only if the new value results in instrument behavior that is the same or better than what you requested. Coercion of values is essential for instrument interchangeability. If specific drivers did not coerce values, you would have to specify values that were valid for the specific instruments you use. If you attempted to use your program with new instruments, the valid values probably would be different.

  For example, the IviDmm class defines an attribute called IVIDMM_ATTR_RANGE, that you use to specify the measurement range. Typically, DMMs have a discrete set of ranges. One DMM might have discrete settings for the range attribute such as 1 V, 10 V, and 100 V. Another DMM might have discrete settings such as 2 V, 20 V, and 200 V. If you use the first instrument, you might set the range attribute to 10.0. If you then attempt to run your program with the second instrument, 10.0 is no longer an acceptable value. The specific driver for the second instrument therefore coerces the value to 20.0.

In other cases, specific drivers coerce the value you specify for an attribute to the value the instrument would coerce the value to. This coercion is necessary in order to implement the state-caching feature of IVI. For example, if a DMM has discrete settings for the range attribute of 1 V, 10 V, and 100 V, it coerces any value between 2 and 9 V to 10 V. Thus, if you set the range to 2 V, then to 7 V, and then to 5 V, the actual range remains at 10 V. To prevent sending redundant commands to the instrument, the specific driver coerces the value internally, sends the coerced value to the instrument, and caches the coerced value. In the example, the specific driver coerces 2 V to 10 V, sends 10 V to the instrument, and caches 10 V. When you set the range to 5 and 7 V, it coerces these values to 10 V, notices that 10 V is the current cache value, and thus does not send any commands to the instrument.

If you enable coercion recording, the class driver keeps a record of each case in which that the specific driver coerces `ViInt32` or `ViReal64` values. You can view the coercion records to understand how the specific driver coerces values that you specify in your program.

If you enable spying, you can use the NI Spy utility to view the coercion information. If you disable spying, you can use the *`CLASSPREFIX`*`_GetNextCoercionRecord` function to retrieve the coercion information programmatically.

Follow the Class Behavior Model

Each IVI class driver defines a *behavior model*. The behavior model describes the relationships between the functions and attributes of the driver and the behavior of the instrument. The behavior model also describes the order of operations for configuring an instrument and controlling instrument operations. Follow the behavior model for each class in order to maximize the possibility of using your program with other instruments.

For example, the IviScope behavior model recommends that you fetch a waveform from the instrument after you have acquired it and before you reconfigure the instrument for the next acquisition. Some oscilloscopes destroy an acquired waveform each time you reconfigure them, so fetching prior to reconfiguration ensures that the acquired waveform is available. Other types of oscilloscope do not destroy the acquired waveform each time you reconfigure them, however, you should follow the fetching recommendation of the behavior model so that your program can fetch acquired waveforms successfully from either type of oscilloscope.

The *IVI Class Driver Online Help* also describes the behavior model for each class.

Use the Default Setup to Configure Instrument-Specific Attributes

For some instruments, you might have to configure instrument-specific attributes for the instrument to behave interchangeably. However, keep in mind that any instrument-specific attributes in your program significantly reduce the likelihood that your program can work with another instrument.

You can specify default values for instrument-specific attributes in the Default Setup tab of the IVI Configuration utility. You can access the Default Setup tab in the properties dialog box of any virtual instrument to specify default values for instrument-specific attributes. The class driver applies the default setup after it invokes one of the LabVIEW or LabWindows/CVI initialization functions. By using the default setup mechanism, you can configure instrument-specific attributes outside of your program.

For information on how to use the IVI Configuration utility to configure a default setup, refer to Chapter 4, *Configuring Your System*.

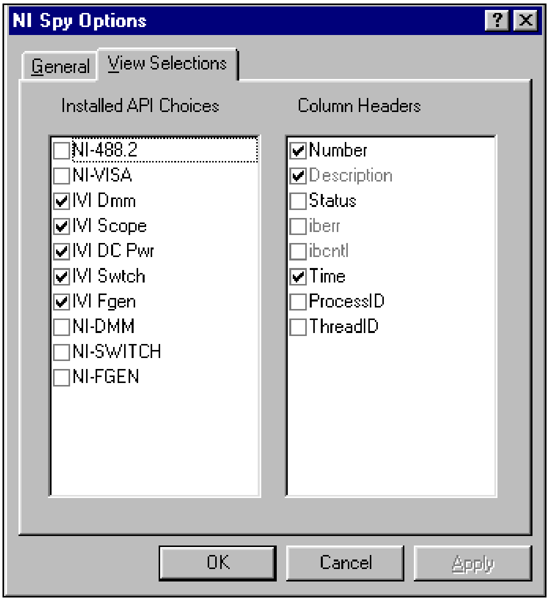
Analyzing Your Program with NI Spy

NI Spy is an application that monitors the use of IVI libraries in Windows applications. It can monitor, record, and display calls made to IVI class drivers and other National Instruments APIs. This section describes how to use the NI Spy utility to monitor calls that your program makes to IVI class drivers.

Configuring NI Spy

To spy on a particular virtual instrument, you must enable spying for that virtual instrument and for the IVI class driver. Refer to Chapter 4, *Configuring Your System*, for a description of how to use the IVI Configuration utility to enable spying for a particular virtual instrument.

You enable spying for an IVI class driver within the NI Spy utility. To launch the NI Spy utility, click the **Start** button in the Windows taskbar and select **Programs»VXIpnp»NI Spy**. To enable spying for a particular class driver, select **Spy»Options**. The class drivers appear in the NI Spy Options dialog box for you to select. By default, all class drivers are selected for spying. Figure 5-6 shows the NI Spy Options dialog box where you make this setting.



Installed API Choices

□ NI-488.2
□ NI-VISA
☑ IVI Dmm
☑ IVI Scope
☑ IVI DC Pwr
☑ IVI Swtch
☑ IVI Fgen
□ NI-DMM
□ NI-SWITCH
□ NI-FGEN

Column Headers

☑ Number
☑ Description
□ Status
□ iberr
□ ibcntl
☑ Time
□ ProcessID
□ ThreadID

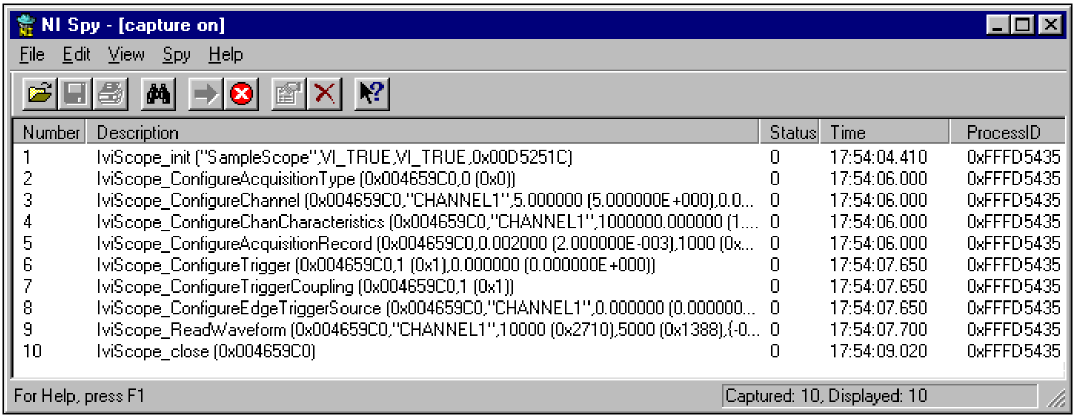
Figure 5-6. Enabling Spying for IVI Class Drivers in NI Spy Options Dialog Box

Capturing Calls to IVI Class Drivers

To view calls to IVI class drivers, you must enable capturing. When you open NI Spy, capturing is disabled. To enable capturing, do one of the following:

- Select **Spy»Start Capture**.
- Click the blue arrow button on the toolbar.
- Press <F8>.

After you enable capturing, run your application and then return to NI Spy to view the captured calls. For a complete description of how to use the NI Spy utility, refer to the NI Spy Windows help. To view the Windows help for NI Spy, select **Help»Help Topics**. Figure 5-7 shows calls to the IviScope class driver that NI Spy captured.



Number	Description	Status	Time	ProcessID
1	IviScope_init ("SampleScope",VI_TRUE,VI_TRUE,0x00D5251C)	0	17:54:04.410	0xFFFD5435
2	IviScope_ConfigureAcquisitionType (0x004659C0,0 (0x0))	0	17:54:06.000	0xFFFD5435
3	IviScope_ConfigureChannel (0x004659C0,"CHANNEL1",5.000000 (5.000000E+000),0.0...	0	17:54:06.000	0xFFFD5435
4	IviScope_ConfigureChanCharacteristics (0x004659C0,"CHANNEL1",1000000.000000 (1....	0	17:54:06.000	0xFFFD5435
5	IviScope_ConfigureAcquisitionRecord (0x004659C0,0.002000 (2.000000E-003),1000 (0x...	0	17:54:06.000	0xFFFD5435
6	IviScope_ConfigureTrigger (0x004659C0,1 (0x1),0.000000 (0.000000E+000))	0	17:54:07.650	0xFFFD5435
7	IviScope_ConfigureTriggerCoupling (0x004659C0,1 (0x1))	0	17:54:07.650	0xFFFD5435
8	IviScope_ConfigureEdgeTriggerSource (0x004659C0,"CHANNEL1",0.000000 (0.000000...	0	17:54:07.650	0xFFFD5435
9	IviScope_ReadWaveform (0x004659C0,"CHANNEL1",10000 (0x2710),5000 (0x1388),{-0...	0	17:54:07.700	0xFFFD5435
10	IviScope_close (0x004659C0)	0	17:54:09.020	0xFFFD5435

For Help, press F1

Captured: 10, Displayed: 10

Figure 5-7. Calls to the IviScope Class Driver

The NI Spy utility displays the name of each class driver function call that it captures. For each function call, the NI Spy utility displays the values of the input and output parameters. You can configure NI Spy to display the following details regarding the function call:

- Number
- Description
- Status
- Time
- Process ID
- Thread ID

If a function returns an error, NI Spy highlights the function call in red. If you enable interchangeability checking, NI-Spy highlights the functions that report interchangeability warnings in blue.

Call Properties

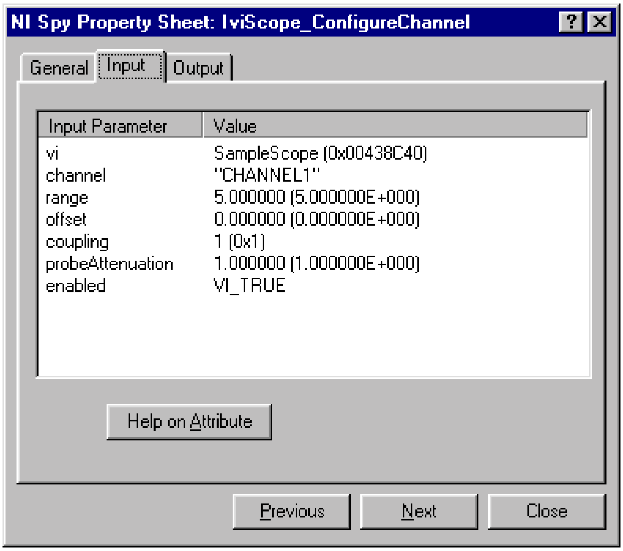
NI Spy records detailed information about each call that it captures. To see the detailed information for a specific call, do one of the following:

- Double-click the call in the capture window.
- Select the call and press <Enter>.

- Select the call and select **View»Properties**.
- Select the call and then click the **Properties** button on the toolbar.

For a complete description of the information that you can view in the NI Spy Property Sheet dialog box, refer to the Windows help for NI Spy. The Property Sheet dialog box contains additional information when it shows a call to an IVI class driver. The remainder of this section describes additional information you can view for calls to IVI class drivers.

Figure 5-8 shows the Input tab of the Property Sheet dialog box.



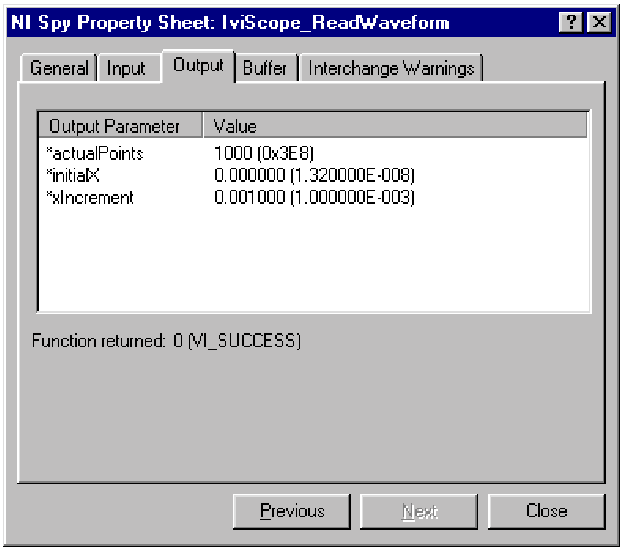
Input Parameter	Value
vi	SampleScope (0x00438C40)
channel	"CHANNEL1"
range	5.000000 (5.000000E+000)
offset	0.000000 (0.000000E+000)
coupling	1 (0x1)
probeAttenuation	1.000000 (1.000000E+000)
enabled	VI_TRUE

Figure 5-8. Input Tab of the NI Spy Property Sheet Dialog Box

The Input tab of the NI Spy Property Sheet dialog box displays the input parameters value for a function call. When you call the class driver initialize function, you pass the logical name of the instrument you want to initialize. The initialize function returns an IVI session handle that identifies the instrument session. You pass the IVI session handle as the `vi` input parameter to all other class driver functions. For functions that have a `vi` input parameter that represents an IVI session handle, NI Spy displays the value of the `vi` parameter and the logical name that corresponds to the value. Thus, you can easily identify the particular instrument session that a call accesses.

The Output tab displays the output parameters and status information for the function call. For calls to an IVI class driver, NI Spy displays additional error information. This information includes the primary error, the secondary error, and any error elaboration information.

Figure 5-9 shows the Output tab of the NI Spy Property Sheet for a call to an IVI class driver.



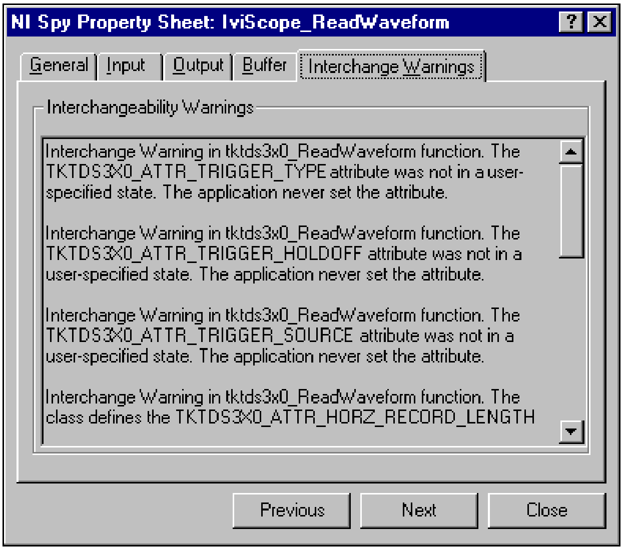
Output Parameter	Value
*actualPoints	1000 (0x3E8)
*initialX	0.000000 (1.320000E-008)
*xIncrement	0.001000 (1.000000E-003)

Function returned: 0 (VI_SUCCESS)

Figure 5-9. Output Tab of the NI Spy Property Sheet Dialog Box

You can use NI Spy to view interchangeability warnings. If you enable interchangeability checking and a class driver function encounters an interchangeability warning, an Interchange Warnings tab appears on the NI Spy Property Sheet dialog box for the function call. The Interchange

Warnings tab displays the interchangeability warnings that the function call produced. Figure 5-10 shows the Interchange Warnings tab of the NI Spy Property Sheet dialog box.


NI Spy Property Sheet: IviScope_ReadWaveform
General
Input
Output
Buffer
Interchange Warnings
Interchangeability Warnings
Previous
Next
Close

Interchange Warning in tktds3x0_ReadWaveform function. The TKTDS3X0_ATTR_TRIGGER_TYPE attribute was not in a user-specified state. The application never set the attribute.

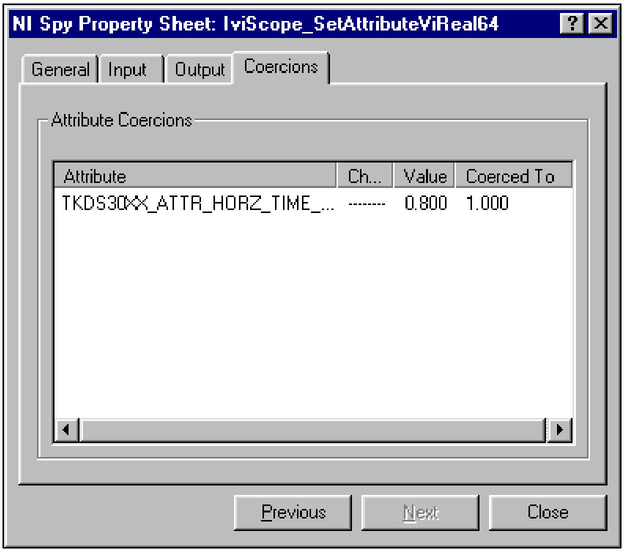
Interchange Warning in tktds3x0_ReadWaveform function. The TKTDS3X0_ATTR_TRIGGER_HOLDOFF attribute was not in a user-specified state. The application never set the attribute.

Interchange Warning in tktds3x0_ReadWaveform function. The TKTDS3X0_ATTR_TRIGGER_SOURCE attribute was not in a user-specified state. The application never set the attribute.

Interchange Warning in tktds3x0_ReadWaveform function. The class defines the TKTDS3X0_ATTR_HORZ_RECORD_LENGTH

Figure 5-10. Interchange Warnings Tab of the NI Spy Property Sheet Dialog Box

You can view how the specific driver coerces `ViInt32` and `ViReal64` values with NI Spy. If you enable coercion recording and the specific driver coerces values that you pass to a class driver function, a Coercions tab appears on the NI Spy Property Sheet dialog box for the function call. Figure 5-11 shows the Coercions tab of the NI Spy Property Sheet dialog box.



Attribute	Ch...	Value	Coerced To
TKDS30XX_ATTR_HORZ_TIME_...	-------	0.800	1.000

Figure 5-11. Coercions Tab of the NI Spy Property Sheet Dialog Box

The Coercions tab displays the name of the attribute that the specific driver coerced, the value that you specified in your program, and the value to which the specific driver coerced the attribute. If the attribute is channel based, the tab also displays the channel name.

Verifying Instrument Replacement Candidates

This section describes techniques you can use to verify whether a new instrument can work with an existing application.

Developing a Reference Program

One technique for verifying instrument replacement candidates is to create a reference program. You create the reference program when you specify the requirements of your test program or when you develop your test program. The reference program uses an IVI class driver to describe programmatically all the instrument configurations and operations that you require.

To verify whether a new instrument can work with your program, you execute the reference program using the specific driver for the new instrument. You can run the reference program with the physical instrument or with simulation enabled. If you can execute the reference program without errors using the new instrument, then it is likely that you can use the new instrument with your existing application. Depending on how many errors and the types of errors the reference program generates, you can gauge the degree to which you must modify your program to make it work with the new instrument.

Testing the New Driver in Simulation Mode

If you do not have a reference program, you can verify a replacement candidate by trying to run your program with the new instrument. If it is not possible to connect to the physical instrument, you can enable simulation.

Often the analysis routines in your test program depend on the data that an instrument returns. If the instrument does not return valid data, the program returns an error and does not fully execute all the statements that control the instrument. If you enable simulation, you can use the class simulation drivers to create simulated data so that you can fully execute your test program.

If you can execute your program using the new instrument, then it is likely that you can replace your existing instrument with the new one.

6

Advanced Class Driver Simulation

This chapter describes how to configure and use the advanced simulation features of the IVI class drivers.

IVI Class Driver Overview

The IVI class drivers implement advanced simulation features by using simulation drivers. The IVI Driver Toolset installs a simulation driver for each IVI class driver. Each simulation driver "plugs in" to the corresponding class driver and performs flexible output data simulation.

Table 6-1 lists the simulation driver files that the IVI Driver Toolset installs.

Table 6-1. IVI Class Simulation Drivers

Class Drivers	Simulation Driver Files
IviScope	`nisscope.dll`, `nisscope.c`, `nisscope.h`, `nisscopu.uir`, `nisscopu.h`
IviDmm	`nisdmm.dll`, `nisdmm.c`, `nisdmm.h`, `nisdmmu.uir`, `nisdmmu.h`
IviFgen	`nisfgen.dll`, `nisfgen.c`, `nisfgen.h`
IviSwtch	`nisswtch.dll`, `nisswtch.c`, `nisswtch.h`
IviDCPwr	`nisdcpwr.dll`, `nisdcpwr.c`, `nisdcpwr.h`, `nisdcpwu.uir`, `nisdcpwu.h`
Common Files	`nisimu.uir`, `nisimu.h`

The IVI Driver Toolset distributes the simulation drivers as `.dll` files so that you can use them immediately. It also includes C source files for the simulation drivers so you can modify the drivers to meet your specific simulation requirements. The IVI Driver Toolset installation program places the simulation driver `.dll` files in the `<`*`vxipnpbase`*`>\bin` directory. The installation program places the `.c`, `.h`, and `.uir` files in the `<`*`vxipnpbase`*`>\niivi\sim` directory.

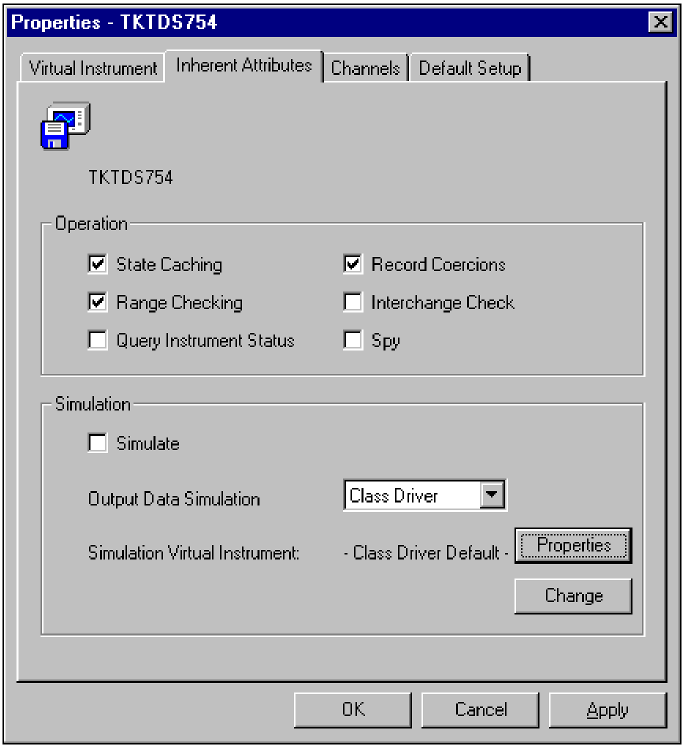
Configuring Simulation

Use the IVI Configuration utility to enable and configure simulation for each logical name in your system. Refer to Chapter 4, *Configuring Your System*, for a complete description of how to use the IVI Configuration utility.

To configure simulation, do the following:

1. Launch the IVI Configuration Utility.
2. Right-click the logical name of the instrument you want to simulate.
3. Select the **Properties** command.
4. Click the **Properties** button for the Virtual Instrument, and then click the **Inherent Attributes** tab.

The Properties dialog box appears as shown in Figure 6-1.



TKTDS754

Operation

☑ State Caching ☑ Record Coercions

☑ Range Checking ☐ Interchange Check

☐ Query Instrument Status ☐ Spy

Simulation

☐ Simulate

Output Data Simulation: Class Driver

Simulation Virtual Instrument: - Class Driver Default -

Figure 6-1. Properties Dialog Box

Use the Simulation controls group to configure how the class driver simulates the specific instrument. To enable simulation, select the Simulate control. Setting the Simulate control has the same effect as setting the `PREFIX_ATTR_SIMULATE` attribute.

It is important to realize that even when you enable simulation, the class driver initializes a session to the specific driver. The specific driver range checks and coerces all input parameters but does not perform instrument I/O. Therefore, you can verify that your program is performing valid operations for the instrument even when the instrument is not available.

The **Output Data Simulation** ring control configures whether the class driver or the specific driver generates the simulation data for output parameters. If you select `Class Driver` from the **Output Data Simulation** ring control, the class driver controls the generation of simulation data for output controls. To do so, the class driver initializes two instrument drivers—the specific driver for the instrument you are simulating and a simulation driver.

The class driver determines which simulation driver to initialize based on the contents of the Simulation Virtual Instrument indicator. When you call a class driver function in your program, the class driver calls the corresponding function in the specific driver and then calls the same function in the simulation driver. The specific driver range checks and coerces all input parameters. The simulation driver generates simulation data for output parameters.

If you select `Specific Driver` from the **Output Data Simulation** ring control, the specific driver generates simulation data for output parameters. In this case, the class driver initializes a session to the specific driver and does *not* initialize a session to the simulation driver.

Configuring the Simulation Virtual Instrument

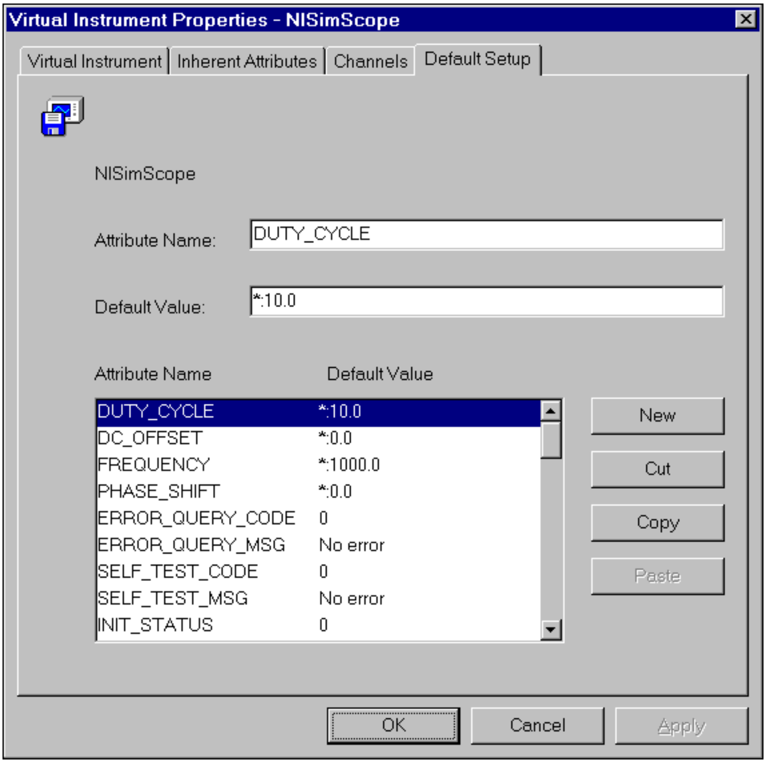
When you configure the class driver to generate output simulation data, the class driver uses the virtual instrument that the Simulation Virtual Instrument indicator displays. The simulation virtual instrument is a logical construct that identifies the simulation driver and its configuration. To change the simulation virtual instrument that the class driver uses, click the **Change** button. The IVI Driver Toolset creates a default simulation virtual instrument for each class. To use the default class simulation virtual instrument, select the **Class Driver Default** option.

Use the default class simulation virtual instrument when you want to configure the simulation in the same way for all instruments of that class.

If you want to configure the simulation differently for each instrument, use the **Change** button to make separate copies of the class default simulation virtual instrument. You can then configure the simulation separately for each instrument in your system.

To change the configuration of the simulation virtual instrument that appears in the Simulation Virtual Instrument indicator, click the **Properties** button. A Properties dialog box for the simulation virtual instrument appears. If you want to specify a different simulation driver, click the Virtual Instrument tab. To configure the behavior of the simulation driver, click the Default Setup tab. In the Default Setup tab, you specify attribute settings that the class driver applies to the simulation driver after initialization. For your convenience, the default simulation virtual instrument that the IVI Driver Toolset creates for each class includes all attributes of the simulation driver.

The Default Setup tab of the Properties dialog box for the IviScope simulation virtual instrument appears in Figure 6-2.



NISimScope

Attribute Name: DUTY_CYCLE

Default Value: *:10.0

Attribute Name	Default Value
DUTY_CYCLE	*:10.0
DC_OFFSET	*:0.0
FREQUENCY	*:1000.0
PHASE_SHIFT	*:0.0
ERROR_QUERY_CODE	0
ERROR_QUERY_MSG	No error
SELF_TEST_CODE	0
SELF_TEST_MSG	No error
INIT_STATUS	0

Figure 6-2. Default Setup Tab, Simulation Virtual Instrument Properties Dialog Box

All simulation drivers have the following attributes you can configure with the Default Setup tab of the Simulation Virtual Instrument Properties dialog box.

Category or Attribute

`PREFIX_ATTR_INTERACTIVE_SIMULATION`
VXI*plug&play* Function Simulation

```
    PREFIX_ATTR_SELF_TEST_CODE
    PREFIX_ATTR_SELF_TEST_MSG
    PREFIX_ATTR_ERROR_QUERY_CODE
    PREFIX_ATTR_ERROR_QUERY_MSG
    PREFIX_ATTR_DRIVER_REV_QUERY
    PREFIX_ATTR_INSTR_REV_QUERY
```

Status Code Simulation

```
    PREFIX_ATTR_SIMULATE_STATUS_CODES
    PREFIX_ATTR_INIT_STATUS
    PREFIX_ATTR_CLOSE_STATUS
    PREFIX_ATTR_RESET_STATUS
    PREFIX_ATTR_SELF_TEST_STATUS
    PREFIX_ATTR_ERROR_QUERY_STATUS
    PREFIX_ATTR_ERROR_MESSAGE_STATUS
    PREFIX_ATTR_REVISION_QUERY_STATUS
    <attributes for each function specific to the IVI
    class>
```

- The *PREFIX*_ATTR_INTERACTIVE_SIMULATION attribute specifies whether you configure the simulation driver with interactive panels. If you set this attribute to True or 1, the simulation driver displays a Simulator Setup dialog box when you call the *ClassPrefix*_init function. With the Simulator Setup dialog box you configure the simulation driver at run time. If you set the *PREFIX*_ATTR_INTERACTIVE_SIMULATION attribute to False or 0, you must configure the simulation driver in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box.
- The VXI*plug&play* Function Simulation category contains attributes that configure data that the *ClassPrefix*_self_test, *ClassPrefix*_error_query, and *ClassPrefix*_revision_query functions return.
- The Status Code Simulation category contains attributes that set the return value of instrument driver functions. Each instrument driver function has an attribute you use to configure a simulated status code for the function. The Status Code Simulation attributes configure

simulated status codes for all the functions that the IVI class drivers export, except for the following functions:
*ClassPrefix*_GetErrorInfo
*ClassPrefix*_ClearErrorInfo
*ClassPrefix*_GetNextInterchangeWarning
*ClassPrefix*_GetNextCoercionRecord
*ClassPrefix*_LockSession
*ClassPrefix*_UnlockSession
*ClassPrefix*_SetAttribute<type>
*ClassPrefix*_GetAttribute<type>
*ClassPrefix*_CheckAttribute<type>

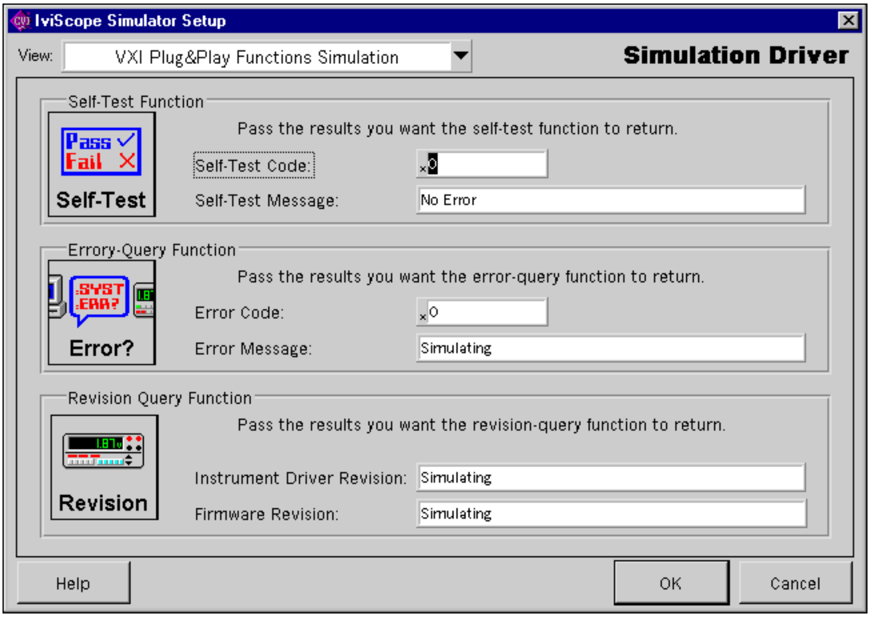
The *PREFIX*_ATTR_SIMULATE_STATUS_CODES attribute specifies whether to simulate status codes. When you set the *PREFIX*_ATTR_SIMULATE_STATUS_CODES attribute to True or 1, the simulation driver has the following behavior. If no other error exists, the simulation driver returns the simulation status code you configure for the function. Simulated errors override existing warnings. Simulated warnings override VI_SUCCESS.

When you set the *PREFIX*_ATTR_SIMULATE_STATUS_CODES attribute to False or 0, the simulation driver does not simulate status codes.

For a complete description of all the simulation driver attributes refer to the *IVI Class Driver Online Help*.

Interactive Simulation

Each class simulation driver has a Simulator Setup dialog box. You can use the Simulator Setup dialog box to configure the simulation driver interactively. If you enable interactive simulation, the simulation driver displays its Simulator Setup dialog box when you call an initialize function. The Simulator Setup dialog box allows you to configure the simulation driver at run time. Everything you can configure in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box is also configurable through the Simulator Setup dialog box. Figure 6-3 shows the IviScope Simulator Setup dialog box.



IviScope Simulator Setup

View: VXI Plug&Play Functions Simulation

Simulation Driver

Self-Test Function

Pass the results you want the self-test function to return.

Self-Test Code: 0

Self-Test Message: No Error

Errory-Query Function

Pass the results you want the error-query function to return.

Error Code: 0

Error Message: Simulating

Revision Query Function

Pass the results you want the revision-query function to return.

Instrument Driver Revision: Simulating

Firmware Revision: Simulating

Figure 6-3. IviScope Simulator Setup Dialog Box

The Simulator Setup dialog boxes for the different class drivers have common features. Each Simulator Setup dialog box has a **View** ring control in the upper left-hand corner. Use the **View** ring control to select a feature of the simulation driver to configure. All the dialog boxes have views to configure the simulation of the VXI*plug&play* functions and the simulation of function status codes. For classes that take measurements, such as IviDmm, IviScope, and IviDCPwr, the Simulator Setup dialog box also contains a Measurement Data Simulation view.

Because you can fully configure the simulation driver in the Simulator Setup dialog box at run time, you do not have to use the IVI Configuration Utility to specify attribute values in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box. If you do specify a default setup, however, the values that you specify appear as the initial values in the Simulator Setup dialog box each time you call `ClassPrefix_init`.

VXI*plug&play* Function Simulation

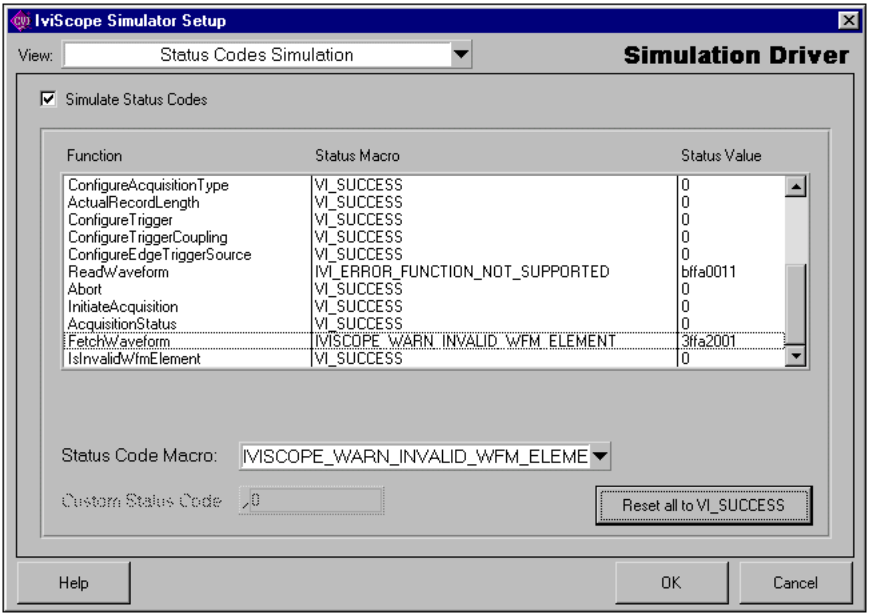
To configure the simulation of the VXI*plug&play* functions, select `VXIplug&play Function Simulation` from the View ring control. Figure 6-3 shows the view you use to configure VXI*plug&play* function simulation. The VXI*plug&play* Function Simulation view is the same for all simulation drivers. The following table describes the controls that appear in the VXI*plug&play* Function Simulation view, including the names of the related attributes:

Table 6-2. Controls in the VXI*plug&play* Function Simulation View

Control Label	Description	Related Attribute in the Default Setup Tab
Self-Test Code	Value that the *`Prefix`*`_self_test` function returns in the self-test code parameter.	*`PREFIX`*`_ATTR_SELF_TEST_CODE`
Self-Test Message	String that the *`Prefix`*`_self_test` function returns in the self-test message parameter.	*`PREFIX`*`_ATTR_SELF_TEST_MSG`
Error Code	Value that the *`Prefix`*`_error_query` function returns in the error code parameter.	*`PREFIX`*`_ATTR_ERROR_QUERY_CODE`
Error Message	String that the *`Prefix`*`_error_query` function returns in the error message parameter.	*`PREFIX`*`_ATTR_ERROR_QUERY_MSG`
Instrument Driver Revision	String that the *`Prefix`*`_revision_query` function returns in the instrument revision parameter.	*`PREFIX`*`_ATTR_DRIVER_REV_QUERY`
Firmware Revision	String that the *`Prefix`*`_revision_query` function returns in the instrument firmware revision parameter.	*`PREFIX`*`_ATTR_INSTR_REV_QUERY`

Status Code Simulation

To configure the status code simulation, select `Status Code Simulation` from the **View** ring control. Figure 6-4 shows the Status Code Simulation view of the IviScope Simulator Setup dialog box.



IviScope Simulator Setup

View: Status Codes Simulation

Simulation Driver

☑ Simulate Status Codes

Function	Status Macro	Status Value
ConfigureAcquisitionType	VI_SUCCESS	0
ActualRecordLength	VI_SUCCESS	0
ConfigureTrigger	VI_SUCCESS	0
ConfigureTriggerCoupling	VI_SUCCESS	0
ConfigureEdgeTriggerSource	VI_SUCCESS	0
ReadWaveform	IVI_ERROR_FUNCTION_NOT_SUPPORTED	bffa0011
Abort	VI_SUCCESS	0
InitiateAcquisition	VI_SUCCESS	0
AcquisitionStatus	VI_SUCCESS	0
FetchWaveform	IVISCOPE_WARN_INVALID_WFM_ELEMENT	3ffa2001
IsInvalidWfmElement	VI_SUCCESS	0

Status Code Macro: IVISCOPE_WARN_INVALID_WFM_ELEME

Custom Status Code: 0

Figure 6-4. Status Code Simulation View

The Status Code Simulation view is the same for all simulation drivers. The Status Code Simulation view has the following controls:

- **Simulate Status Codes**—Specifies whether to simulate status codes for the instrument driver functions. Setting this control has the same effect as setting the *`PREFIX`*`_ATTR_SIMULATE_STATUS_CODES` attribute in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box.
- **Entries**—Is a list box that contains each function that the class driver defines and the simulation status macro and code that you specify for the function. When you select an entry in the list box, the contents of the entry appear in the controls below the list box.

- **Status Code Macro**—Is a ring control from which you can select a status code macro. Each class driver defines custom status codes and status codes that are common to all class drivers. The class drivers define a unique macro for each status code. The Status Code Macro ring control contains all the status code macros for the particular class driver you are using. With the Status Code Macro ring control you can easily identify and select a particular status code to simulate.

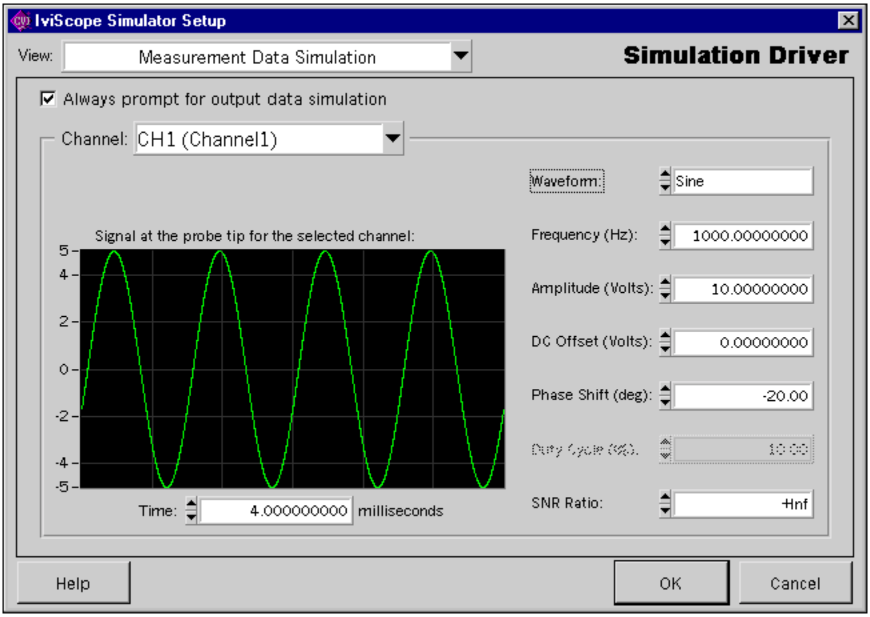
  If you select Custom Status Code, you configure a custom status code in the Custom Status Code control.
- **Custom Status Code**—Lets you specify a custom status code to simulate for a particular function.
- **Reset all to `VI_SUCCESS` button**—Sets the status code macro to `VI_SUCCESS` and the status value to `0` for all functions.

Configuring simulated status codes with this dialog box has the same effect as setting the corresponding simulated status code attributes in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box.

Measurement Data Simulation

The IviDmm, IviScope, and IviDCPwr simulation drivers perform measurement data simulation. For example, when you use the IviScope class driver with simulation enabled, you can configure the waveform that the `IviScope_ReadWaveform` and `IviScope_FetchWaveform` functions return.

Figure 6-5 shows the Measurement Data Simulation view for the IviScope simulation driver.



IviScope Simulator Setup

View: Measurement Data Simulation

Simulation Driver

☑ Always prompt for output data simulation

Channel: CH1 (Channel1)

Waveform: Sine

Frequency (Hz): 1000.00000000

Amplitude (Volts): 10.00000000

DC Offset (Volts): 0.00000000

Phase Shift (deg): -20.00

Duty Cycle (%): 10.00

SNR Ratio: +Inf

Time: 4.000000000 milliseconds

Figure 6-5. IviScope Measurement Data Simulation View

The measurement data simulation view for each simulation driver typically has an **Always Prompt for Output Data Simulation** checkbox control. You use the **Always Prompt for Output Data Simulation** control to specify whether you want the simulation driver to display the Measurement Data Simulation panel each time your program takes a measurement. If you enable the Always Prompt for Output Data Simulation control, you can configure separately the data that the simulation driver generates for each measurement. When you disable the Always Prompt for Output Data Simulation control, you configure the output data simulation once for the instrument session when you call an initialization function.

You can configure the measurement data simulation in the Default Setup tab of the Simulation Virtual Instrument Properties dialog box. For a complete description of measurement data simulation for the IviScope, IviDmm, and IviDCPwr simulation drivers, refer to the *IVI Class Driver Online Help*.

Non-Interactive Simulation

When you disable interactive simulation, you can configure simulation only in the IVI Configuration utility. After you initialize your driver by calling an initialization function, you cannot alter the configuration of the simulation driver. Non-interactive simulation is useful when you want to prevent the interactive panels from interrupting your test program.

Advanced Simulation Topics

This section describes advanced class driver simulation topics.

Modifying the Simulation Driver

The IVI class simulation drivers provide general purpose simulation features. However, you might require more application-specific simulation capabilities. For that reason, the IVI Driver Toolset includes C source code for the simulation drivers. You can customize the data simulation algorithms for your specific requirements. Because simulation drivers work with the class drivers, you can reuse the simulation code you develop with different specific instruments.

If you want to modify the user interface panels for the simulation driver, you must install LabWindows/CVI.

Note If you modify the simulation drivers, you must not change the simulation driver function prototypes. If you change the prototypes, your driver may give unpredictable results.

User-Interface Requirements

The interactive capabilities of simulation drivers place additional requirements on your system. The simulation driver `.dll` files that ship with the IVI Driver Toolset require the LabWindows/CVI Run-time Engine. The IVI Driver Toolset installation program installs the LabWindows/CVI Run-time Engine.

If you want to deploy a simulation driver on a system that does not have the LabWindows/CVI Run-time Engine, you must modify and recompile the simulation driver source code. This section describes the modifications you must make for both LabWindows/CVI 5.0.*x* and for LabWindows/CVI 5.5.

LabWindows/CVI 5.0.*x*—To modify and recompile the simulation driver using the LabWindows/CVI 5.0.*x* compiler, perform the following steps:

1. Edit the `.c` file of the simulation driver. The `.c` file for each simulation driver contains the following statements:

   ```
   #ifndef ALLOW_INTERACTIVE_SIMULATION
   #define ALLOW_INTERACTIVE_SIMULATION
   ```

   To prevent your program from trying to invoke the LabWindows/CVI Run-time Engine, change the `ALLOW_INTERACTIVE_SIMULATION` macro definition from `1` to `0`. That way, the compiler does not process any of the user interface code.
2. Enable the **Instrument Driver Support Only** command in the **Build** menu of the Project window. When you enable the **Instrument Driver Support Only** command, your project does not link to the LabWindows CVI Run-time Engine. To enable the **Instrument Driver Support Only** command, select **Build»Instrument Driver Support Only**.
3. Select the **Build»Create Dynamic Link Library** command in the Project window to compile the simulation driver `.dll` file. The Create Dynamic Link Library dialog box appears. Be sure to export the symbols from the `.h` file of the simulation driver as follows:
   a. Click the **Change** button to display the DLL Export Options dialog.
   b. Select `Include File Symbols` from the **Export What** control
   c. Select the `.h` file for the simulation driver in the **Which Project Include Files** list control.
4. Click the **OK** button to create the DLL.

LabWindows/CVI 5.5—To modify and recompile the simulation driver using the LabWindows/CVI 5.5 compiler, perform the following steps:

1. Edit the `.c` file of the simulation driver. The `.c` file for each simulation driver contains the following statements:

   ```
   #ifndef ALLOW_INTERACTIVE_SIMULATION
   #define ALLOW_INTERACTIVE_SIMULATION
   ```

   To prevent your program from trying to invoke the LabWindows/CVI Run-time Engine, change the `ALLOW_INTERACTIVE_SIMULATION` macro definition from `1` to `0`. That way, the compiler does not process any of the user interface code.

2. Select **Build»Target Settings** in the Project window. The Target Settings dialog appears. Enable the **Instrument Driver Support Only** control if you do not want to link to the LabWindows/CVI Run-time Engine. Be sure to export the symbols from the `.h` file of the simulation driver, as follows:
   d. Click the **Change** button in the export section to display the DLL Export Options dialog box.
   e. Select `Include File Symbols` from the **Export What** control
   f. Select the `.h` file for the simulation driver in the **Which Project Include Files** list control.
3. Select **Build»Create Release Dynamic Link Library** in the Project window to compile the simulation `.dll` file.

After you compile the simulation driver, the resulting `.dll` does not require the CVI Run-time Engine, but you cannot use the interactive simulation panels. If you attempt to enable interactive simulation, the simulation driver returns an error.

Enabling Simulation after Initializing

If you have disabled simulation when you initialize the instrument driver, you can enable simulation at a later time. When you disable simulation at the time of initialization, the IVI engine sets the driver to perform instrument I/O, therefore the instrument must be present in your system. You can then enable or disable simulation at any time by setting the *PREFIX*_ATTR_SIMULATE attribute. If you configure the simulation driver for interactive simulation, the simulation driver displays the Simulator Setup dialog box the first time you enable simulation. When you enable simulation, the simulation driver behaves as this chapter describes. This approach is useful if you want to simulate the instrument only during specific times that you run your application.

A

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `www.ni.com`

NI Developer Zone

The NI Developer Zone at `zone.ni.com` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `www.ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `www.ni.com`

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `www.ni.com`. Branch office web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

A

ADE

Application Development Environment

B

behavior model

Defines the relationships between instrument driver attributes and functions with instrument behavior.

C

class driver

An instrument driver that provides a generic programming interface to instruments of a particular class. From your test program, you make calls to a class driver, which in turn communicates through a specific driver for your instrument. You can change the specific instrument driver (and corresponding instrument) in your system underneath the class driver without affecting your test code.

control

An input and/or output device that appears on a function panel that allows a user to specify function parameters or that displays function results.

F

`.fp` file

A file that contains information that allow the interactive program to display function panels that correspond to a specific instrument driver.

function panel

A user interface in LabWindows/CVI libraries that allows you to interactively execute library functions and to generate code for insertion in a program.

function tree
The hierarchical structure that defines the grouping of functions in an instrument driver.

H

handle
Session identifier for your IVI program that you use to call drivers.

I

include file
A file that contains function declarations, constant definitions, and external declaration of global variables that the instrument driver exports.

initialize
In IVI, obtain a handle for a session to an instrument driver, using one of the IVI Initialization functions.

initiate
Cause an instrument to begin an operation, using one of the IVI Initiate functions.

instrument driver
A set of routines designed to control an instrument, and a set of data structures to represent the driver.

instrument handle
See handle.

Instrument library
A LabWindows/CVI library that contains instrument drivers.

IVI Configuration utility
A section within the National Instruments Measurement & Automation Explorer that you use to configure your IVI applications.

IVI inherent attributes
Attributes that all IVI specific drivers and IVI class drivers must implement.

IVI refnum
Session identifier for your IVI program that you use to call drivers.

K

ksamples
1,000 samples

L

LabVIEW

ADE for graphical programming

LabWindows/CVI

ADE for programming in ANSI C

logical name

The name that you pass to identify the particular virtual instrument that the IVI engine uses when you call a class driver initialize function. The virtual instrument, in turn, identifies a particular specific driver and device and specifies the initial settings for the session. If you want to use your program with a different physical instrument, you change the properties of the logical name to use the virtual instrument for the new instrument. You use the IVI Configuration utility to create and edit logical names.

M

MB

Megabytes of memory

N

NI Spy

An application monitor for Windows applications that use National Instruments drivers. It can monitor, record, and display calls made to IVI class drivers. NI Spy quickly locates any erroneous calls your application makes to the IVI class drivers.

R

refnum

See IVI refnum.

S

session

The instance of an object that is created when you initialize the instrument driver. The session ends when you close the driver. The sessions are used to ensure safe behavior of multiple driver instances in multi-instrument applications.

simulation drivers

A set of routines that IVI class drivers use to implement advanced simulation features. The IVI Driver Toolset installs a simulation driver for each IVI class driver. Each simulation driver “plugs in” to the corresponding class driver and performs flexible simulation of data output.

simulation virtual instrument

A logical construct that identifies the simulation driver and its configuration. Simulation virtual instruments reference simulation drivers instead of specific drivers. They do not reference a device.

specific driver

A high-level function library that you use to control a specific GPIB, VXI, or serial instrument or other device. A specific driver contains the information required to control a particular model of instrument, including the command strings, parsing code, and valid ranges of each setting for that particular instrument.

V

V

volts

value parameter

A scalar parameter of integer, single-precision, long, or double-precision data type whose value is not modified by the subroutine or function. In other words, an integer, single-precision, long, or double-precision scalar parameter is a value parameter if and only if its function panel control is *not* an output control.

virtual instrument (VI)

A program in the graphical programming language G that models the appearance and function of a physical instrument. In the IVI Driver Toolset, a virtual instrument is an item that you configure with the IVI Configuration utility. It identifies a particular specific driver and device and specifies the initial settings for the session. To change the initial settings for the session, you modify the properties of the virtual instrument.

Index