

National Instruments IVI[™] Driver Help

January 2007 Edition, Part Number 370430C-01

This help file is intended for IVI Compliance Package software users.

Refer to <u>IVI Compliance Package</u> for more information about this NI software package.

To navigate this help file, use the **Contents, Search,** and **Index** tabs to the left of this window.

For more information about this help file, refer to the following topics:

Using Help

Related Documentation

<u>Glossary</u>

Important Information

Technical Support and Professional Services

To comment on National Instruments documentation, refer to the <u>National</u> <u>Instruments Web site</u>.

© 2007 National Instruments Corporation. All rights reserved.

Using Help

<u>Conventions</u> <u>Navigating Help</u> <u>Searching Help</u> <u>Printing Help File Topics</u>

Conventions

This help file uses the following conventions:

- < > Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>.
- [] Square brackets enclose optional items—for example, [response].
- 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.
- The symbol indicates that the following text applies only to a specific product, a specific operating system, or a specific software version.
- This icon denotes a note, which alerts you to important information.
- This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
- **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, emphasis, or an introduction to a key concept.
- **dark red** Text in this color denotes a caution.
- green Underlined text in this color denotes a link to a help topic, help file, or Web address.
- *italic* Italic text denotes variables or cross references. 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 Bold text in this font denotes the messages and responses
 bold that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

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

Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- Hide—Hides the navigation pane from view.
- Locate—Locates the currently displayed topic in the Contents tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- **Forward**—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

Wildcards

You also can search using asterisk (*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



Note Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

Nested Expressions

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

Boolean Expressions

Click the **•** button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

Search Options

Use the following checkboxes on the **Search** tab to customize a search:

- Search previous results—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- Match similar words—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- Search titles only—Searches only in the titles of topics.

Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

- 1. Right-click the book.
- 2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
- 3. Select the **Print the selected heading and all subtopics** option.
 - Note Select Print the selected topic if you want to print the single topic you have selected in the **Contents** tab.
- 4. Click the **OK** button.

Printing PDF Documents

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

Related Documentation

You might find the following documentation helpful as you use this help file:

- <u>Application Notes</u>—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.
- <u>Measurement & Automation Explorer Help for IVI</u>—
 This help file discusses how to configure your system with MAX.
- LabWindows/CVI Instrument Driver Developers Guide—This document describes guidelines for writing an IVI instrument driver. To download and use this document, click <u>ni.com/manuals</u>, and search for the LabWindows/CVI Instrument Driver Developers Guide.
- <u>IVI Foundation</u>—This Web site provides information from the IVI Foundation, including the specifications for each instrument class.

Glossary



Numbers/Symbols

80	Infinity.
1D	One-dimensional.
2D	Two-dimensional.
Δ	Delta; difference.
ΔX	denotes the value by which <i>x</i> changes from one index to the next.
П	Pi.
Α	
A	Amperes.
A/D	Analog-to-digital; analog/digital.
absoluto coordinatos	Picture coordinates relative to the origin

(0,0) of the picture indicator.

absolute coordinates

absolute path	File or directory path that describes the location relative to the top level of the file system.
AC	Alternating current.
active window	Window that is currently set to accept user input, usually the frontmost window. The titlebar of an active window is highlighted. Make a window active by clicking it or by selecting it from the Windows menu.
ADE	Application Development Environment. An environment that provides users various software development tools for the development of automated test systems. Examples of ADEs include National Instruments LabVIEW and National Instruments LabWindows/CVI.
AI	Analog input.
analog trigger	Trigger that occurs at a user-selected level and slope on an incoming analog signal. You can set triggering to occur at a specified voltage on either an increasing or a decreasing signal (positive or negative slope).
ANSI	American National Standards Institute.
API	Application Programming Interface.
Application Programming Interface (API)	A standardized set of subroutines or functions along with the parameters that a program can call.
array	Ordered, indexed list of data elements of the same type.
ASCII	American Standard Code for Information Interchange.
B	

The minimum set of functions, attributes, and attribute values that an IVI driver must

implement to claim compliance with an instrument class specification. For example, the base class capabilities of the oscilloscope class have functions and base class capabilities attributes that configure an edge-triggered acquisition, initiate an acquisition, and return the acquired waveform. An IVI class-compliant specific driver implements all the base capabilities for a particular class. A diagram indicating an instrument's possible states, the functions that cause it behavior model to transition between states, and the attributes that affect tis behavior in each state. С A set of functions, attributes, and attribute values defined in an IVI Foundation specification. There are four different capability group types of instrument capabilities—Inherent Capabilities, Base Class Capabilities, Class Extension Capabilities, and instrument specific driver Capabilities. An instrument specific driver string that refers to a particular channel of a device. An IVI specific driver that implements channel string channels defines the channel strings that it recognizes. A channel string is an example of a repeated capability identifier. One of multiple physical inputs or outputs to an instrument. A set of channels is a channel type of repeated capability. 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	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. Groups of functions, attributes, and
class extension capabilities	attribute values that an instrument class specification defines to represent instrument class features that are more specialized than the features that the Base Class Capabilities represent. A driver implements an extension capability only if the instrument being controlled by the driver supports the specialized features of the particular extension capability group. For example, the IviScope specification defines extension capability groups for various trigger modes, such as glitch triggering and TV triggering. IVI class-compliant specific instrument drivers are not required to implement extension groups, but should implement all class extensions that the instrument hardware supports. For a complete description of the class extension capabilities for a particular class, refer to individual class specifications. Refer Section 2.6, <i>Capability Groups</i> , of <i>IVI 3.1: Driver</i> <i>Architecture Specification</i> , for more information on class extension capabilities at ivifoundation.org.
coercion	Refer to <u>value coercion</u> .
Context Help window	Special window in IVI Soft Front Panels that describes the function or action associated with the underlying control.

D

DLL	Dynamic Link Library
driver	A software module that controls a hardware device. Also refer to IVI driver.
driver session	A session for an IVI driver.
driver session configuration	An item you configure in MAX to associate an IVI driver with initial settings and possibly one or more hardware assets.
E	
error code	A value returned by the instrument driver when execution does not successfully complete.
extension capability	Refer to class extension capabilities.
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.
G	
GPIB	General Purpose Interface Bus. The common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987.

Η

handle	A unique identification that allows access to a session.
hardware asset	A physical device or instrument that performs a measurement or stimulus function.
hardware configuration attribute	An attribute that allows the user to set and retrieve the value of an instrument setting.
I	
I/O resource descriptor	Refer to resource descriptor.
include file	A file that contains function declarations, constant definitions, and external declaration of global variables that the instrument driver exports.
initiate	Cause an instrument to begin an operation using one of the IVI Initiate functions.
inherent capabilities	The set of functions, attributes, and attribute values that all IVI drivers are required to implement.
instrument	A type of hardware asset.
instrument class	A way to typify instruments. Instruments with common capabilities can be considered to be of the same class. For example, "Function Generator" is an instrument class typified by an ability to generate voltage signals.
instrument driver	A set of routines designed to control an instrument, and a set of data structures to represent the driver.
instrument handle	Refer to <u>handle</u> .

The ability to exchange hardware assets

instrument interchangeability	in a test system with few or no changes to the test program. Functions, attributes, and attribute values that represent features not defined by an instrument class specification. For example, some oscilloscopes have special features such as jitter and timing
instrument specific driver capabilities	IviScope class specification. The functions, attributes, and attribute values necessary to access the jitter and timing analysis capabilities of an oscilloscope are considered instrument specific driver capabilities. Instrument specific driver capabilities are beyond the scope of an instrument class.
IVI	Interchangeable Virtual Instruments
IVI class driver	An IVI driver that exposes a class- compliant API and serves as a pass- through layer to IVI class compliant specific drivers. For example, an IviScope class driver exports the functions, attributes, and attribute values defined in the IviScope class specification. When an application program calls an IviScope class driver, the IviScope class driver calls an IVI class-compliant specific driver that communicates with an oscilloscope. IVI class drivers are necessary for instrument interchangeability when using IVI-C class- compliant specific drivers.
IVI class-compliant specific driver	An IVI specific driver that complies with one of the IVI class specifications. For example, an IVI class-compliant specific driver for an oscilloscope exports the API defined by the IviScope class specification. Typically, an IVI class-
	compliant specific driver also provides

	instrument specific driver capabilities.
IVI class specification	A specification that defines the standard set of interfaces for an instrument class.
IVI driver	hardware device and that complies with the IVI Foundation specifications. For IVI Foundation compliance requirements, refer to <i>IVI-3.1: Driver Architecture</i> <i>Specification</i> at ivifoundation.org.
IVI driver installer	An IVI installer that installs IVI drivers.
IVI engine	A support library for IVI instrument drivers that performs common tasks such as session creation, attribute management, and instrument status checking.
IVI Library	Refer to IVI engine. The terms IVI engine and IVI Library are used interchangeably throughout IVI driver documentation and refer to the same software component.
IVI specific driver	An IVI driver that contains information for controlling a particular instrument or family of instruments and communicates directly with the instrument hardware. For example, IVI specific drivers control message-based instrument hardware by sending command strings and parsing responses.
L	
LabVIEW LabWindows/CVI	Laboratory Virtual Instrument Engineering Workbench—a program development application based on the programming language G and used commonly for test and measurement purposes. ADE for programming in ANSI C The name that you pass to identify the

logical name	engine uses when you call a class driver initialize function. The driver session, in turn, identifies a particular specific driver and hardware asset 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 driver session for the new instrument. You use MAX to create and edit logical names.
Μ	
MAX	Refer to <u>Measurement & Automation</u> Explorer.
Measurement & Automation Explorer	A controlled, centralized configuration environment that allows you to configure all your National Instruments devices.
Ν	, ,
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.
physical identifier	Refer to <u>physical repeated capability</u> identifier.
physical repeated capability identifier	An instrument specific driver string that refers to a particular instance of a repeated capability. An IVI specific driver that implements repeated capabilities defines the repeated capability identifiers that it recognizes.

published API	An API that is designed for implementation across several software modules, and as such is published independently of any one software module that implements it. Examples are IVI instrument class APIs and IVI-MSS role APIs.
R	
range checking	The functionality of a driver that validates parameter and attribute values against published instrument limits. Range checking is a configurable behavior that can be enabled or disabled by the user. An instrument capability for which multiple
repeated capability	instances can be configured independently. Examples of repeated capabilities include channels on an oscilloscope, traces on a spectrum analyzer, and modulators on a RF signal generator.
repeated capability identifier	A virtual or physical repeated capability identifier.
repeated capability name	The name published by an IVI driver or class specification to refer to a particular repeated capability. For example, the IviScope class specification defines "Channel" as a repeated capability name. A string, such as a VISA resource
resource descriptor	descriptor, that specifies the I/O address of a hardware asset.
resource name S	Refer to <u>resource descriptor</u> .

A run-time instance that provides context

session	for communicating and interacting with a particular physical instrument or collection of physical instruments. For IVI drivers, a session maintains configuration information and instrument state information from one IVI driver call to another and across threads. The user creates a driver session by calling the Initialize function of the driver. When creating a session in an application program, the user may reference a session configuration from the IVI configuration store. The session ends when you close the driver.
session configuration	An item you configure in MAX to associate an instrument driver with initial settings and possibly one or more hardware assets.
simulation	A required feature of IVI drivers that allows users to develop application code even when the instrument is not available. In simulation mode, the instrument driver does not perform I/O on the instrument.
simulation drivers	A set of routines that IVI class drivers use to implement simulation features. The IVI Compliance Package 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 driver session	A logical construct that identifies the simulation driver and its configuration. Simulation driver sessions reference simulation drivers instead of specific drivers. Simulation driver sessions do not reference a hardware asset.
specific driver	Refer to <u>IVI specific driver</u> . A behavior of IVI specific drivers that

state caching	tracks the state of instrument settings at run time. When the user enables state caching on an IVI driver that implements state caching, the instrument driver performs instrument I/O only when the current state of an instrument setting is different from what the user requests. State caching can improve performance of a test program by preventing the driver from sending redundant commands to the instrument.
status code	An error code or completion code.
V	
V	volts
value coercion	Occurs when an IVI specific driver alters the value that the user specifies for an attribute or parameter to a value that the specific driver or instrument can accept. If the specific driver coerces an attribute value, the specific driver returns the coerced value when the user reads the value of the attribute. For attributes that represent a continuous range of values, a driver may coerce the value that the user requests to a value that is more appropriate for the instrument.
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 identifier	Refer to <u>virtual repeated capability</u> identifier.

virtual repeated capability identifier	An alias the user defines in a driver session in the IVI configuration store to represent a physical repeated capability identifier. The IVI configuration store contains the mappings between virtual repeated capability identifiers and physical repeated capability identifiers. Users striving for instrument interchangeability should use virtual repeated capability identifiers in their application programs.
VI Tree	The hierarchical structure that defines the grouping of VIs in an instrument driver.
VISA	Virtual Instrument Software Architecture. VISA provides an I/O API for communicating over a variety of bus interfaces, including GPIB, VXI, and serial interfaces.
VPP	VXIplug&play.
VXI	VMEbus Extensions for Instrumentation, or IEEE 1155.VME eXtensions for Instrumentation.
VXI <i>plug&play</i> Systems Alliance	An organization whose members share a common commitment to end-user success with open, multivendor test and measurement systems. To view specifications defined by this alliance, refer to vxipnp.org.
W	
warning code	A value that an instrument driver returns after successful execution to provide additional information to the user. For example, if an instrument does not support a self test operation, calling the self test function will return a self test not supported code.

NI IVI Compliance Package

The NI IVI Compliance Package is a software package that contains IVI class drivers and the support libraries necessary for the development and use of applications that leverage IVI instrument interchangeability.

The IVI Compliance Package is also based on and is compliant with the latest version of the instrument programming specifications defined by the IVI Foundation. The IVI class drivers in the ICP are used with IVI specific drivers. You can either download IVI specific drivers from ni.com/idnet or create them yourself.

What the Setup Program Installs

The setup program for the IVI Compliance Package installs the following components on your hard disk:

- IVI Class Drivers
- IVI Engine
- Measurement & Automation Explorer (MAX)
- IVI Shared Components

IVI Class Drivers

The IVI Compliance Package supports eight driver classes: oscilloscope, digital multimeter, function/arbitrary waveform generator, DC power supply, switch, RF signal generator, spectrum analyzer, and power meter. You can develop hardware independent test programs with these driver classes. The setup program installs the files you need to use the class drivers with LabWindows/CVI, LabVIEW, and Measurement Studio.

LabWindows/CVI

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 includes file (.h), which contains the function declarations and constant definitions for the class.
- The class driver function panel file (.fp), which 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.

LabVIEW

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 documents the hierarchy of VIs for the class driver.
- A Windows help file (.hlp), which contains documentation for the LabVIEW class driver.

MeasurementStudio

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

• The static libraries for various run-time library compatibilities.

- The class driver includes files (.h and .inl), which contain the class and constant definitions for the class.
- The following figure shows how a class driver redirects function calls from a test program to the correct specific driver.
- The component .xml file, which allows the class to be used with MeasurementStudio.
- A Microsoft Help file (.chm), which contains documentation for the MeasurementStudio class driver.

Refer to <u>IVI Class Drivers Overview</u> for more information about class drivers.

IVI Engine

The IVI engine is the key support library that enables IVI Instrument Drivers to perform simulation and other performance enhancing benefits, such as state caching, range checking and status checking. During development, these features can aid in development of your application. Once you transition into production mode, you'll want to achieve peak performance. The IVI engine allows you to disable features like range checking and status checking, thus allowing you to achieve maximum production throughput.

IVI Shared Components

The IVI Shared Components are owned and distributed by the IVI Foundation. These components make the underlying structure of an IVIcompliant system. Since IVI Compliance Package components are designed to work with the IVI Shared Components, the IVI Shared Components are installed as part of the IVI Compliance Package.

The Configuration Server included in the IVI Shared Components provides an API for accessing instrument driver data. Various IVI products have been redesigned to work with the Configuration Server instead of the previous way of using the ivi.ini file.

IVI Driver Help Introduction

The topics in this book are intended for both LabWindows/CVI and LabVIEW users. These topics describe how to develop hardware independent test programs with IVI (*Interchangeable Virtual Instruments*) instrument drivers.

These topics also give 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 these topics when you develop, debug and deploy test programs that use IVI instrument drivers.

Instrument Driver Overview

What Is an Instrument Driver?

Programmers used to drive computercontrolled instrumentation systems with 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 GPIB. Each instrument responded to particular ASCII strings as documented in each vendor's instrument user manual. Programmers were responsible for learning each command set and writing the control program.

Now programmers use highlevel routines that hide the lowlevel commands. Also, by using routines that are generic and modular, programmers can reuse them in future applications that use the same instrument. These reusable routines are known as instrument drivers.

An instrument driver is a highlevel function library that you use to control a specific GPIB, VXI, or serial instrument or other device. With an instrument driver, you can easily control an instrument without knowing the lowlevel command syntax or I/O protocol.

Historical Evolution of Instrument Drivers

Early instrument driver 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 open, modifiable drivers built around standards that allowed instruments from a variety of vendors to easily coexist in one application.

The VXI*plug&play* Systems Alliance improved existing instrument driver standards and enabled system interoperability. Users could install VXI*plug&play* instrument drivers from a variety of vendors on the same system without encountering system conflicts. In addition, these standards used VISAdefined data types to define parameters of all instrument driver functions. These data types promoted 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 (specifically, state caching), and test development flexibility (specifically, range checking and simulation).
The IVI Foundation

The IVI Foundation is an organization of endusers, 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 Interchangeable Virtual Instruments (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. Some of the most important features are as follows:

- Hardware Independence—IVI class instrument drivers allow developers to build systems that reuse existing test programs with different instruments. The benefits of instrument interchangeability extend to a wide variety of applications, including the following examples:
 - Test system developers in the military and aerospace industries, who must maintain test systems and code for many years, can easily reuse test code on new equipment as instruments improve or become obsolete.
 - Manufacturers in competitive, highvolume industries, such as telecommunications and consumer electronics, can keep their production lines running when instruments malfunction or must be recalibrated.
 - Large companies can easily reuse and share test code between departments and remote sites without being required to use the same instrumentation hardware.
- Instrument State Caching—Standard VXI*plug&play* drivers do not keep track of the instrument state. 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 difference seems minor in approach, but it can lead to reductions in test time and cost.

• Instrument Simulation—IVI drivers can simulate the operation of an instrument when that instrument is not available. Developers can enable simulation in IVI drivers to create simulated data for output parameters. With simulated data, developers can develop stable code for instruments even when the instruments are not available.

The IVI Foundation identifies instrument classes based on common functionality in test equipment. That common functionality is used to define flexible Application Programming Interfaces (APIs) to meet the needs of test system developers. Each API is referred to as a class, and its implementation a class driver.

For example, the IviScope class contains functionality that is common to most oscilloscopes, such as vertical range, offset, timebase, trigger mode, waveform acquisition, and so on. The class uses functions and attributes to provide access to all of the included features. Test system developers use the class drivers to write software that works with any oscilloscope, regardless of communication bus or manufacturer.

The IVI Foundation also defines a common driver architecture and other requirements to further benefit test system developers. All NI class drivers conform to IVI architecture and class specifications.

For more information about the IVI Foundation, visit *ivifoundation.org*.

Historical Evolution of Instrument Drivers

Although the instrument driver concept 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 open and modifiable drivers, 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; that is, state caching, and test development flexibility; specifically, range checking and simulation.

Advantages of IVI Drivers Over Traditional Instrument Drivers

IVI drivers have many advantages:

- 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 drivers.
- National Instruments can more easily maintain and upgrade instrument drivers when each driver has only one set of source files. Simpler maintenance 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.

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 companies can easily reuse and share test code between departments and remote sites without being required 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 can lead to reductions in test time and cost.

• Instrument Simulation—IVI drivers can simulate the operation of an instrument when that instrument is not available. Developers can enable simulation in IVI drivers to create simulated data for output parameters. With simulated data, developers can develop stable code for instruments even when the instruments are not available.

Refer to IVI System Architecture for a comprehensive list of IVI features.

IVI Instrument Specific Drivers Overview

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 help file 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 prevents the specific driver from sending redundant commands to the instrument. If you try to set an attribute to a value that is already configured, the specific driver skips sending the command. You can disable state caching.
- **Range Checking**—Range checking verifies that a value you specify for an attribute is within the valid range for the instrument, without performing costly I/O or causing an instrument error. You can disable this feature for faster execution speed.
- **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 for an instrument driver even when the instrument is not available. In simulation mode, the instrument driver generates simulated data for output parameters.
- **Multithread Safety**—You can use the IVI instrument driver in multithreaded applications. Multiple execution threads can use the same IVI instrument session without interfering with each other.

IVI Class Drivers Overview

IVI class drivers implement functions and attributes for controlling an instrument within a specified class, as defined by the IVI Foundation.

Each IVI class driver consists of generic code that can call IVI instrument specific drivers. Through configuration you can switch between various specific drivers, achieving interchangeability. By using IVI class drivers in your test program, you can change hardware without changing test code or recompiling your application.

The following figure shows how a class driver redirects function calls from a test program to the correct specific driver.



The IVI Foundation manages the definition of the instrument classes. The IVI Foundations 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.

Interchangeability

Interchangeability of IVI class drivers depends on the fundamental interchangeability of the hardware that you are using. Your test system requirements still dictate your choice of particular instruments. For example, if your test system requires DMM measurements with $8\frac{1}{2}$ digits of precision, you must use a DMM with $8\frac{1}{2}$ digits of precision. You cannot replace an $8\frac{1}{2}$ digit DMM with a $5\frac{1}{2}$ digit DMM in your test system, unless you require only $5\frac{1}{2}$ digits of precision, regardless of the software architecture. IVI class drivers implement a standard architecture for swapping instruments that are capable of taking the required measurements for your test system.

Using IVI Instrument Drivers

In general, you operate and develop test programs with IVI instrument drivers the same way as traditional instrument drivers.

IVI drivers are of a significantly higher quality than traditional drivers and use state caching to optimize performance. IVI drivers are highly configurable and you can optimize an IVI driver without modifying the driver source code.

LabWindows/CVI

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

If you are using LabWindows/CVI, refer to <u>IVI Class Driver Help for</u> <u>LabWindows/CVI</u> for information on how to use instrument drivers with LabWindows/CVI.

LabVIEW

Use the .llb, .rc, and set of .mnu files when you use an IVI driver in LabVIEW. 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 a complete G source code like native LabVIEW drivers.

Refer to <u>IVI Class Driver Help for LabVIEW</u> for more information about using instrument drivers in LabVIEW.

IVI System Architecture

Main Components of an IVI System

The following figure 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 contains all the information to control a particular instrument model.



IVI System Architecture

Your test program communicates with the specific driver in one of two ways: directly or through an IVI class driver.

Directly

Your test program can call the specific driver directly, similar to the way you use traditional instrument drivers. With this approach, you gain most of the benefits of IVI instrument drivers, including state caching, range checking, 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 specific driver's prefix is unique, interchangeability is not available through direct communication, and you must modify and recompile your test program when you want to use a different specific driver.

Through an IVI Class Driver

Your test program can access a specific driver indirectly through an IVI class driver. With IVI class drivers, you can develop test programs that are independent of specific hardware and interchange instruments without modifying or recompiling your test program. In addition to interchangeability, IVI class drivers deliver other benefits such as class simulation, spying, and interchangeability checking. For information about a particular IVI class driver, refer to <u>IVI Class Driver Help</u>.

When you use an IVI class driver, you begin by calling its initialize function. This function uses the logical name parameter to initialize the instrument and driver software. The function then returns an instrument handle that you use with all other IVI class driver function calls. You create and configure logical names in MAX.

To change the instrument that your program uses, edit the logical name in MAX 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 features in MAX, refer to <u>Measurement & Automation Explorer Help for IVI</u>.

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.

The names of all component files (.fp, .h, .dll, .llb, and so on) of the class driver begin with the class prefix.

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.*
Power Meter	IviPwrMeter	IVIPWRMETER_	ivipwrmeter.*
RF Signal Generator	IviRFSigGen	IVIRFSIGGEN_	ivirfsiggen.*
Spectrum Anaylzer	IviSpecAn	IVISPECAN_	ivispecan.*

Class Capability Groups

Because many instruments of a given class have additional, different functionality, no single programming interface can work with all instruments. For this reason, the IVI class drivers divide the instrument capabilities into the following capability categories:

- Inherent IVI Capabilities
- Base Capabilities
- Extension Groups
- Instrument Specific Capabilities

Inherent IVI Capabilities

Inherent IVI Capabilities are the functions and attributes that all IVI class drivers implement.



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 DMM class *ClassPrefix* Initialize VI becomes IviDmm Initialize VI in LabVIEW, and *ClassPrefix_*init becomes IviDMM_init in LabWindows/CVI.

Use the following links to browse the tables, which show the inherent IVI VIs/functions and properties/attributes.

Inherent IVI VIs/Functions

- Initialize/Close
- Get, Set, and Check Property/Attribute
- <u>Utility</u>
- <u>Subcategories (Error Info,</u> <u>Interchangeability Info, Coercion Info,</u> <u>and Locking)</u>

Inherent IVI Properties/Attributes

- User Options
- <u>Class Driver</u> Identification
- <u>Specific Driver</u>
 <u>Identification</u>
- <u>Specific Driver</u> <u>Capabilities</u> <u>Category</u>
- Instrument
 Identification
- Error Info
- <u>Advanced</u>
 <u>Session</u>
 <u>Information</u>

Inherent IVI VIs/Functions

LabVIEW Inherent VI	LabWindows/CVI Inherent Function
Initialize/Close—Contains VIs and instrument driver sessions.	functions that initialize and close
ClassPrefix Initialize	ClassPrefix_init
ClassPrefix Initialize With Options	ClassPrefix_InitWithOptions
ClassPrefix Close	<i>ClassPrefix</i> _close
Set, Get, and Check Attribute/Property —Contains VIs and functions that 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>.</type>	
Use Property Node VI to get and	ClassPrefix_SetAttribute <type></type>
set attributes. You can not check	<i>ClassPrefix_</i> GetAttribute <type></type>
	<i>ClassPrefix_</i> CheckAttribute <type></type>
	ClassPrefix_InvalidateAllAttributes
ClassPrefix Invalidate All Attributes	

Utility—VIs and functions that 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.

ClassPrefix Reset	ClassPrefix_reset
ClassPrefix Self-Test	ClassPrefix_self_test
ClassPrefix Revision Query	<i>ClassPrefix_</i> revision_query

ClassPrefix Error-Query	ClassPrefix_error_query
ClassPrefix Error Message	ClassPrefix_error_message
ClassPrefix Reset With Defaults	ClassPrefix_ResetWithDefaults
ClassPrefix Disable	ClassPrefix_Disable
	ClassPrefix_GetSpecificDriverCHandle

Error Info Subcategory		
Use the General Error Handler VI	ClassPrefix_GetError	
to view errors, which is the standard LabVIEW approach.	<i>ClassPrefix_</i> ClearError	

Interchangeability Info Subcateg	jory
ClassPrefix Get Next Interchange	ClassPrefix_GetNextInterchangeWarning
Warning	ClassPrefix_ResetInterchangeCheck
<i>ClassPrefix</i> Reset Interchange Check	<i>ClassPrefix</i> _ClearInterchangeWarnings
<i>ClassPrefix</i> Clear Interchange Warnings	

Coercion Info Subcategory	
<i>ClassPrefix</i> Get Next Coercion Record	ClassPrefix_GetNextCoercionRecord

Locking Subcategory	
You cannot lock IVI sessions in	ClassPrefix_LockSession
LabVIEW.	ClassPrefix_UnlockSession

Note Use VIs and properties when you configure IVI drivers in LabVIEW. Use functions and attributes when you configure IVI



drivers in LabWindows/CVI.

The following table shows the inherent IVI properties and attributes.

Inherent IVI Properties/Attributes

LabVIEW Property	LabWindows/CVI Attribute
User Option specific drive	\mathbf{s} —Attributes that you can set to affect the behavior of class rs.
Range Check	CLASSPREFIX_ATTR_RANGE_CHECK
Query Instrument Status	CLASSPREFIX_ATTR_QUERY_INSTRUMENT_STATUS
Cache	CLASSPREFIX_ATTR_CACHE
Simulate	CLASSPREFIX_ATTR_SIMULATE
Use Specific Simulation	CLASSPREFIX_ATTR_USE_SPECIFIC_SIMULATION
Record Value Coercions	CLASSPREFIX_ATTR_RECORD_COERCIONS
Interchange Check	CLASSPREFIX_ATTR_INTERCHANGE_CHECK
SPY	CLASSPREFIX_ATTR_SPY
Class Driver the class driv	Identification —Attributes that describe the version, supplie rer.
Class Driver Description	CLASSPREFIX_ATTR_CLASS_DRIVER_DESCRIPTION
Class Driver Prefix	CLASSPREFIX_ATTR_CLASS_DRIVER_PREFIX
Class Driver Vendor	CLASSPREFIX_ATTR_CLASS_DRIVER_VENDOR
Class Driver Revision	CLASSPREFIX_ATTR_CLASS_DRIVER_REVISION
Class Driver Class Specification	CLASSPREFIX_ATTR_CLASS_DRIVER_CLASS_SPEC_MAJ

Major Version	
Class Driver Class Specification Minor Version	CLASSPREFIX_ATTR_CLASS_DRIVER_CLASS_SPEC_MIN
Specific Driv and prefix of	ver Identification —Attributes that describe the version, supp the specific driver.
Specific Driver Description	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_DESCRIPTION
Specific Driver Prefix	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_PREFIX
Specific Driver Locator	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_LOCATOR
Specific Driver Vendor	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_VENDOR
Specific Driver Revision	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_REVISION
Specific Driver Class Specification Major Version	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_M
Specific Driver Class Specification Minor Version	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_M
Specific Driv capabilities o	ver Capabilities Category —Attributes that describe various f the instrument controlled by the instrument driver.

Supported Instrument Models	CLASSPREFIX_ATTR_SUPPORTED_INSTRUMENT_MODEL
Group Capabilities	CLASSPREFIX_ATTR_GROUP_CAPABILITIES
Function Capabilities	CLASSPREFIX_ATTR_FUNCTION_CAPABILITIES
Channel Count	CLASSPREFIX_ATTR_CHANNEL_COUNT

Instrument Identification— Attributes that provide model and firmware information about the instrument controlled by the instrument driver.

Manufacturer	CLASSPREFIX_ATTR_INSTRUMENT_MANUFACTURER
Model	CLASSPREFIX_ATTR_INSTRUMENT_MODEL
Firmware Revision	CLASSPREFIX_ATTR_INSTRUMENT_FIRMWARE_REVISI

Advanced Session Information—Attributes that provide resource and I/O session information regarding the class and specific driver.

Logical Name	CLASSPREFIX_ATTR_LOGICAL_NAME
Resource Descriptor	CLASSPREFIX_ATTR_IO_RESOURCE_DESCRIPTOR
Driver Setup	CLASSPREFIX_ATTR_DRIVER_SETUP

Base Capabilities

Base capabilities are the functions and attributes of an instrument class that are common to most of the instruments in that class. A class driver should cover 95% 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 more information about the base capabilities for a particular class, refer to:

- IviDCPwrBase Capability Group
- IviDmmBase Capability Group
- IviFgenBase Capability Group
- IviScopeBase Capability Group
- IviSwtchBase Capability Group

Extension Groups

Extension groups contain specialized groups of functions and attributes that are common to many instruments in that class, but not all. The IVI specifications do not require specific instrument drivers to implement extension groups. With extension groups, the IVI class drivers create standard programming interfaces for features and capabilities that are not common to every instrument of that class.

For example, although all oscilloscopes have similar base capabilities for vertical and horizontal settings, they have a wide variety of trigger modes. The IviScope class driver includes extensions for different trigger modes, such as TV trigger, runt trigger, width trigger, and so on. Every oscilloscope 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 support the TV trigger extension group can still comply with the IviScope class because the oscilloscope can support the base capabilities of the IviScope class.

If you use an extension group in your program, any instrument you use with the program must support the extension group. The LabWindows/CVI function panels, LabVIEW VIs, and help file for the class drivers mark functions and attributes that are a part of an extension group with a special symbol. For example, the IviScope class driver help file marks the functions and attributes that control the TV trigger extension group with the symbol [TV]. For more information about the extension groups of a particular class, refer to:

- IviDCPwr Class Driver Overview
- IviDmm Class Driver Overview
- IviFgen Class Driver Overview
- IviScope Class Driver Overview
- IviSwtch Class Driver Overview

Instrument Specific Capabilities

In addition to inherent IVI capabilities, base instrument capabilities, and extension groups, IVI specific drivers may export instrument specific functions and attributes. These features are usually device-specific functions 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, instead of using a class driver. For information about the instrument specific features of a driver, refer to <u>IVI Instrument Specific Drivers</u>.

Your test program can access both the class driver and the specific driver during execution. In LabVIEW, the current instructions work. In LabWindows/CVI or Measurement Studio, you must use the GetSepecificDriver C Handle to get the specific driver session, then use the new handle to access the specific driver directly.

When you initialize an instrument through a class driver, you can specify default values for instrument specific attributes. MAX allows you to set the initial values of the instrument specific attributes to protect interchangeability.

Class Driver Relationship to Instrument Specific Drivers

The following figure shows an example of the relationship between the capability groups of a class driver and a specific driver. In this figure, 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, or .obj 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.

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 attributes of the specific driver directly.

Your test program can access the inherent IVI, base instrument, and extension capabilities by invoking functions and attributes in either the class driver or specific driver. However, unless you are using a family driver, you must use the class driver to achieve interchangeability.



Note To access the capabilities of the instrument through the

class driver, call the initialize function in the class driver to create the instrument driver session.

In the figure above, 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 classonly functions and attributes.

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, replace *CLASSPREFIX* with one of the class prefixes: IviDmm, IviDCPwr, IviFgen, IviPwrMeter, IviScope, IviSpecAn, IviSwtch or IviRFSigGen. For example, in the DMM class, *CLASSPREFIX_*ATTR_CLASS_DRIVER_VENDOR becomes IVIDMM_ATTR_CLASS_DRIVER_VENDOR.

Description	Function Name
Specific Driver Module Pathname	CLASSPREFIX_ATTR_SPECIFIC_DRIVER_LOCATOR
Class Revision	CLASSPREFIX_ATTR_CLASS_DRIVER_REVISION
Class Specification Major Version	CLASSPREFIX_ATTR_CLASS_DRIVER_SPEC_MAJOR_VER
Class Specification Minor Version	CLASSPREFIX_ATTR_CLASS_DRIVER_SPEC_MINOR_VER
Class Driver Vendor	CLASSPREFIX_ATTR_CLASS_DRIVER_VENDOR
Class Driver	CLASSPREFIX_ATTR_CLASS_DRIVER_DESCRIPTION

Inherent IVI Attributes: Class Driver Access Only

Description	
Spy	CLASSPREFIX_ATTR_SPY
Use Specific Simulation	CLASSPREFIX_ATTR_USE_SPECIFIC_SIMULATION
Function Capabilities	CLASSPREFIX_ATTR_FUNCTION_CAPABILITIES

Initial Settings (formerly Default Setup)

In Measurement & Automation Explorer 3.0, Default Setup was replaced by a new component: Initial Settings. The driver developer publishes these settings.

Refer to National Instruments IVI Driver Help»Configuring Your System for more information about Initial Settings.

Enabling Instrument Simulation

You can simulate an instrument 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; range checking and parameter coercion; and simulated output data.

Instrument Driver Calls

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 indicating that the instrument is not present.

You can use this simulation capability with IVI class drivers and specific drivers, simulating calls to a class driver and to instrument specific functions in the specific driver.

You can enable simulation through MAX. 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 or that you wish to use simulation mode.

Range Checking and Parameter Coercion

You can range check and coerce all input parameters. Every time you try to send a value to an instrument to configure a setting, the driver ensures that the value is valid for the particular setting on that instrument. If necessary, the specific driver coerces the value to an acceptable one for the instrument. The range checking and coercion operations happen within the specific driver software. Even when you write test code without connecting the instrument, the specific driver ensures that each value you attempt to send to the 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 uses one of the following methods to generate simulated data:

- **Specific Driver Simulation**—Each specific driver has basic builtin algorithms to simulate data generation. For example, when using a DMM-specific driver in simulation mode, the Read function may return a random number within the valid range of the DMM's current mode. When you use an oscilloscope specific driver in simulation mode, the Read Waveform function may return a simple sinewave array of data within valid ranges for the oscilloscope. This simple data generation process returns data values to the program variables so that subsequent function calls do not fail for lack of data. However, a simple or random data may not be meaningful.
- User-Defined Simulation—Because you have access to the specific driver source code, you can modify the driver, adding your own data generation algorithms to generate simulated data that more closely applies to the unit under test (UUT) or the application on which you are working. However, your code is only useful 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 that will work regardless of which specific driver is used, the IVI class drivers provide simulation tools.
Class Driver Simulation

The IVI class drivers provide simulation tools. You can use these tools to create simulated output data. The class drivers provide simulation by using a simulation driver. Simulation drivers are components that plug into a class driver. The class driver uses the simulation driver to generate data.

For example, ICP includes five simulation drivers—oscilloscope, DMM, function generator, switch, and DC power supply. Each of these simulation drivers communicates with the corresponding class driver to perform more flexible data generation than in instrument specific drivers. The following diagram illustrates how a class driver uses a class simulation driver.



Class Simulation Driver

When your program calls 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 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. The specific driver performs range checking and coercion on all input parameters, and 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 MAX. 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 the following figure.

🛄 I viD	mm Simulator Setup		×
View:	Measurement Data Simulation	 Simulation D)river
प	Always prompt for ouput data simulation Measurement Simulation Parameters:		
	Force Over-Range		
	Measurement Base	Measurement Noise (+/-)	
	10.00000	1.00000	
н	elp	ок с	ancel

From the panel, you can select a base measurement value and an offset. For example, in the previous figure, 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 non-interactive 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 class simulation features to generate simulated results for the Self-Test, Error Query, and Revision Query functions that all IVI drivers export. You can also use the class simulation tools to generate simulated completion codes for the instrument driver functions. 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, use attributes in the simulation driver. When you use non-interactive simulation, you configure the attributes of the simulation driver with MAX without modifying your

test program code.

ICP includes C source code for the class simulation drivers. You can develop robust simulated data generation algorithms for your test systems and incorporate them into the simulation drivers. Whenever you swap to a different instrument, you can reuse your developed simulation code because the simulation drivers work with the class drivers.

Disabling Unused Extensions

If you use an instrument that has extended capabilities but your application does not configure the settings for the extended capabilities, the settings are in an unknown state. The unknown state could affect the behavior of the instrument capabilities that the program does use. Furthermore, the unknown state is likely to vary from one instrument to another. The settings are likely to be the power-on settings for the instrument. The power-on settings vary from instrument to instrument.

To avoid having different behavior when used with different instruments, each instrument specific driver disables unused extensions by setting the extensions to an interchangeable state. This interchangeable state renders the extension group unable to affect the behavior of the instrument. Normally, an extension remains disabled until your program explicitly uses it. At which point the instrument driver does not have to take any other action. Therefore, if your program sets any values of the extension group, the specific driver does not enable the extension group.

For example, the IviDmm base capabilities control DMMs that 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 sues only the IviDmm base capabilities with an instrument that implements the multipoint extension group, the IviDmm-compliant specific driver sets the multipoint extension group attributes to an interchangeable state when you call either Initiate or Read functions for IviDmm. In addition to these functions, the driver also disables all extensions in the Initialize and Reset With Defaults functions.

To disable the multipoint extension group to the interchangeable state, the IviDmm-compliant specific 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 behavior of the instrument. 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.

For more information about the interchangeable state that the class drivers apply for disabling unused extensions, refer to <u>Interchangeability</u> <u>Checking</u>.

Interchangeability Checking

IVI drivers have a feature called interchangeability checking. Interchangeablility checking verifies that your program produces the same behavior when used with a different instrument.

Enable interchangeability checking by completing one of the following procedures:

- Refer to <u>Configuring Your System In MAX</u> to enable interchangeability checking in MAX.
- 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 interchangeability checking is enabled, the driver queues interchangeability warnings when it encounters instrument configurations that might not produce the same behavior when you use a different instrument. Use <u>NI Spy</u> to view these warnings or call the Get Next Interchange Warning function.

Interchangeability Checking Rules

Note Although interchangeability checking can be performed through the specific driver interface, interchangeability and thus interchangeability checking is only relevant when called through the class driver interface.

The following rules govern interchangeability checking in IVI.

1. An interchangeability check occurs when you invoke a driver operation that depends on the current state of the instrument. For example, IviDmm compliant drivers perform interchangeability checking when your program calls any of the functions shown in the following table.

Name in LabWindows/CVI	Name in LabVIEW
Prefix_Initiate	Prefix Initiate
Prefix_Read	Prefix Read
Prefix_ReadMultiPoint	Prefix Read MultiPoint

IviDmm Functions that Invoke Interchangeability Checking

- 2. The driver performs interchangeability checking on a capability group basis. When you enable interchangeability checking, the driver always performs interchangeability checking on the base capabilities group.
- 3. The 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 driver does not perform interchangeability checking on that group.

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

• **Unspecified State**—An attribute that is in a state that you did not specify. An attribute ends up in an unspecified state if the attribute is not configured by your program 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 unknown. 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–defined attribute to an instrument specific value. Many class attribute values represent a set of discrete settings. For these attributes, the class specifications defines the possible values that you can assign to the attribute. Specific drivers can define additional, instrument specific values for the attribute. When your program sets an attribute to an instrument specific value, your program is likely to behave differently when you run it with different instruments.

For example, the attribute that configures the measurement function can have both class values and instrument specific values. The class 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 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 defines read-only attributes that specific drivers might implement as read/write.

The attributes that return the aperture time in the IviDmm class 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 other class-compliant drivers implement the attribute as read–only. Therefore, if your program sets an attribute that a class defines as read–only, your program may not work with other instruments.

• Value Not Configured—The driver encounters an error when it tries to apply a value to an extension attribute that your program never configures. <u>Disabling Unused Extensions</u> describes how the driver sets the attributes of extension groups that do not have

specified values—it sets the attributes to an interchangeable state. The driver does this to make your program behave the same regardless of whether the instruments you use implement the extension group. Other instruments that implement the extension group might not support the value to which the driver attempts to set the attribute. In this case, the driver queues an interchangeability warning instead of returning an error from the function.

An example of this behavior is the attribute that configures the interpolation method in the IviScope class. If your program does not set the value of this attribute and the specific driver implements the attribute, the driver attempts to set the interpolation method to Sin(x)/x. However, some oscilloscopes always interpolate data points. For these cases, the driver generates an interchangeability warning to indicate that the attribute that controls the interpolation method is not in an interchangeable state.

Each IVI specific driver defines exceptions to the interchangeability checking rules and defines which functions perform interchangeability checking. Refer to <u>IVI Class Driver Help</u> for more information regarding the interchangeability checking rules for a particular class.

Viewing Interchangeability Warnings

You can use <u>NI Spy</u> to view interchangeability warnings. NI Spy highlights functions in blue that have interchangeability warnings.

Alternatively, you can use the Get Next Interchange Warning VI or function to retrieve interchangeability warnings programmatically.



Note If you set the interchangeability checking attribute to True, you must retrieve interchange warnings either through NI Spy or by calling the Get Next Interchange Warning VI or function.

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.

Configuring Your System

Expand this book for topics about configuring IVI instruments using <u>Measurement & Automation Explorer (MAX)</u>.

IVI Class Driver Operation

This book contains books about using IVI class drivers to develop interchangeable applications. This book 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 MAX, you can 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 IVI Class Drivers in LabVIEW

Using IVI Class Drivers in LabWindows/CVI

Using Class Drivers in LabVIEW

Use IVI class drivers in LabVIEW 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**»Instrument I/O»IVI Class Drivers, as shown in the following figure.



Each IVI class driver has a subpalette that 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.

The following figure shows a code example that uses the IviDmm class driver to configure a DMM and take a measurement.



The following figure shows how you can use the property node to access individual 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. Expand the *Visa Functions* book in the *LabVIEW Help* or refer to the *NI-VISA Programmer Reference Manual* for more information on VISA properties and attributes. Refer to the LabVIEW Help for more information about how to use the property node.



Using Class Drivers in LabWindows/CVI

You use the IVI class drivers in the LabWindows/CVI 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, complete the following steps:

- 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.

The following figure shows the Instrument menu after you load the class drivers.



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. Class drivers that are loaded through the Instrument menu are not automatically loaded to your project and can be unloaded at any time. However, you must reload the driver every time you launch LabWindows/CVI.

To add the IVI class drivers into your project, complete one of the following steps:

- 1. Open the Function Panel window.
- 2. Select File»Add to Project.

or

- 1. From the Project window, select Edit»Add to Project»Instrument (.fp).
- 2. Choose the .fp file for the driver.

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 highlevel functions, you can easily initialize and close the instrument, configure the instrument, control instrument operations, and retrieve measurements. High-level 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, the following figure shows the Set Attribute function panel for the IviDmm class driver. Refer to the <u>IVI Class Driver Help for LabWindows/CVI</u> for more information on how to use function panels in LabWindows/CVI.

🔡 IviDmm Class Driver - Set Attribute Vilnt32		
<u>F</u> ile <u>C</u> ode <u>V</u> iew <u>I</u> nstrument <u>L</u> ibrary <u>T</u> ools <u>W</u> indow <u>O</u> ptions <u>H</u> elp		
<u>****</u> •***		
IviDmm_SetAttributeViInt32		
Channel Name Attribute ID		
Attribute Value		
Press <enter> for a list of value constants.</enter>		
Instrument Handle	Status	
dmmHandle		
<pre>IviDmm_SetAttributeViInt32 (dmmHandle, "", IVIDMM_ATTR_FUNCTION,</pre>		
	_	

Developing an Instrument Independent Application

The following links describe items to consider when you use IVI class drivers to develop an instrument-independent test program. Follow the guidelines contained in the following links to maximize the potential for your application to work with other instruments.

- Using Logical Names
- Naming Virtual Channels
- <u>Using High-Level Configuration Functions Rather than Setting</u>
 <u>Individual Attributes</u>
- <u>Minimizing the Use of Extension Capability Groups</u>
- <u>Completely Specifying the State of the Instrument</u>
- <u>Using the Development Mode Settings for Inherent Attributes</u>
- Following the Class Behavior Model
- Using MAX to Configure Instrument Specific Attributes

Using Logical Names

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

For example, the logical name DMM fits the context of an application in which you swap two or more digital multimeters such as a Fluke 45 and an HP 34401A. In contrast, the logical names FL45 and HP34401A do not fit the context because they are too specific.

When you create a logical name, consider if you want to use it in multiple applications. One approach is to create one logical name for all the applications in your system. Another approach is to create a unique logical name for each application.

If you create a global logical name, you can quickly reconfigure your entire system. When replacing an instrument, you reconfigure only one logical name in MAX. The disadvantage of this method is that any change to the configuration of the logical name in MAX affects all applications in your system that use that name. You are also limited by the requirements of all the applications that use the logical name. This approach also requires that you coordinate your applications to 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. However, if you swap an instrument that many applications use, it may take you longer to reconfigure your system. Also, 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.

Naming Virtual Channels

Use virtual channel names to identify a particular channel of an instrument without using instrument specific channel strings. 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. Using the high-level functions to configure an instrument is a good practice. The high-level functions group the setting of related attributes into a single operation. When you call a high-level 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 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 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 an 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.

Minimizing the Use of Extension Capability Groups

Use only the extension capability groups that your test program requires. Minimizing the number of extension capability groups you access for an IVI session maximizes the number of instruments that you can use with your application.

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

The other capability groups are <u>extension capability groups</u>. 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 Specifying 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 do not specify the state of the instrument completely, you considerably increase 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 <u>capability group</u>, 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 <u>base capabilities</u>. After you access a particular <u>extension</u> <u>capability group</u>, you must configure all attributes of that extension group. Normally, you configure the attributes through one or more high-level configuration functions.

Note 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, you do not have to specify the state of the second attribute. However, NI recommends that you specify the state of all attributes within the capability groups that you use for the sake of long-term stability and reuse.

For example, 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.

Using the Development Mode Settings for Inherent Attributes

IVI class drivers have many features that help you to 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, enable the production mode settings for the inherent attributes to maximize performance.

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

Development Mode and Production Mode Settings for Inheren	t
Attributes	

LabVIEW Property	Inherent Attribute	Dev Moc
<i>ClassPrefix</i> Spy	CLASSPREFIX_ATTR_SPY	V
<i>ClassPrefix</i> Range Check	CLASSPREFIX_ATTR_RANGE_CHECK	V]
<i>ClassPrefix</i> Query Instrument Status	CLASSPREFIX_ATTR_QUERY_INSTRUMENT_STATUS	V]
<i>ClassPrefix</i> Interchange Check	CLASSPREFIX_ATTR_INTERCHANGE_CHECK	V]
<i>ClassPrefix</i> Record Value Coercions	CLASSPREFIX_ATTR_RECORD_COERCIONS	V]

Refer to <u>Measurement & Automation Explorer Help for IVI</u> for information about how to use MAX to set the development mode and production mode settings for inherent attributes.

CLASSPREFIX_ATTR_RANGE_CHECK

Enables/disables range checking. If you enable range checking, the specific driver checks all parameters passed to the class driver and specific driver functions.

CLASSPREFIX_ATTR_QUERY_INSTRUMENT_STATUS

Enables/disables instrument status checking. If you enable instrument status checking, the specific driver checks the status of the instrument after most calls made 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 driver queues warnings when it encounters instrument configurations that are not likely to produce the same behavior when you use a different instrument. The class driver obtains interchangeability warnings from the specific driver, so you can find additional documentation on the warnings provided in the specific driver's help file.

Note You can use <u>NI Spy</u> to view interchangeability warnings. NI Spy highlights driver functions that report interchangeability warnings in blue. If you do not have NI Spy, you can use the PREFIX_GetNextInterchangeWarning function to retrieve interchangeability warnings programmatically.

CLASSPREFIX_ATTR_RECORD_COERCIONS

Enables/disables coercion recording. In many cases, specific drivers coerce the value that you specified for a function parameter or attribute to a value that 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 valid values for the specific instruments you use. If you attempted to use your program with new instruments, the valid values would probably be different.

For example, the lviDmm 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, so the specific driver for the second instrument therefore coerces the value to 20.0.

In other cases, specific drivers coerce the value you specified for an attribute to the value the instrument would coerce the value to. This coercion is necessary to implement the IVI state caching feature. 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 V 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 this 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 V and 7 V, the driver coerces these values to 10 V, notices that 10 V is the current cache value, and does not send any commands to the instrument.

If you enable coercion recording, the driver keeps a record of each instance a ViInt32 or ViReal64 value is coerced. You can view the coercion records to understand how the specific driver coerces values that you specified in your program.

Note You can use NI Spy to view the coercion information. If you do not have NI Spy, you can use the *PREFIX_*GetNextCoercionRecord function to retrieve the coercion information programmatically.

Following 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 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 oscilloscopes 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.

LabVIEW Behavior Model	LabWindows/CVI Behavior Model
IviDCPwr	IviDCPwr
<u>lviDmm</u>	<u>lviDmm</u>
<u>lviFgen</u>	<u>lviFgen</u>
<u>lviScope</u>	<u>IviScope</u>
IviSwtch	IviSwtch

Using MAX to Configure Instrument Specific Attributes

Refer to National Instruments IVI Driver Help»Configuring Your System Using MAX for more information about using Initial Settings to configure instrument specific attributes.

Analyzing Your Program with NI Spy

NI Spy is an application monitor for Windows applications using NI drivers. NI Spy can <u>monitor, record, and display calls</u> 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. NI Spy captures all function calls made during a class driver session.

To spy on IVI class driver calls, the NI Spy attribute must be enabled on the class driver session. This attribute is enabled by default. You can programmatically disable the attribute by setting it to FALSE.

📸 NI Spy	- [capture on]			_ 🗆 🗙
<u>F</u> ile <u>E</u> dit	<u>V</u> iew <u>S</u> py <u>H</u> elp			
	≝ m >⊗ m× m			
Number	Description	Status	Time	
1	<pre>IviScope_init ("SampleScope", VI_TRUE, VI_TRUE, 0x007A003E)</pre>	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	<pre>IviScope_ConfigureTrigger (SampleScope, 1, 0.000000E+000)</pre>	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, pr	ess F1 Ca	ptured: 7, Display	yed: 7	1.

The following figure shows a sample trace from NI Spy.

NI Spy records all input parameters passed 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. The following figure shows a sample property sheet.

Spy Property Sheet: IviScope_ReadWaveform 🛛 🔋 🗙				
General Input Ou	General Input Output Buffer Interchange Warnings			
Input Parameter vi channel waveformSize maxTime &waveform &actualPoints &initia⊠ &shintia⊠	Value SampleScope (0x00438C40) "CHANNEL1" 1000 (0x3E8) 5000 (0x1388) 0x008A404D 0x008A0078 0x008A007E 0x008A0088			
Help on <u>A</u> ttribute				
	Previous <u>N</u> ext (Close		

The property sheets display detailed information such as:

- 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

Configuring NI Spy

To spy on a particular driver session, you must enable spying for that driver session and for the IVI class driver. Refer to <u>Measurement &</u> <u>Automation Explorer Help for IVI</u> for more information about how to use MAX to enable spying for a particular driver session.

To enable spying for an IVI class driver within NI Spy, launch NI Spy by clicking **Start»Programs»National Instruments»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. The following figure shows the NI Spy Options dialog box where you enable this setting.


Capturing Calls to IVI Class Drivers

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

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

After you enable capturing, run your application then return to NI Spy to view the captured calls. For a complete description of how to use NI Spy, refer to the *NI Spy Windows Help*. To view the NI Spy help, select **Help»Help Topics**. The following figure shows calls to the IviScope class driver that NI Spy captured.

📸 NI Sp	y - [capture on]			_ 🗆 ×	
<u>File</u> <u>E</u> di	<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>S</u> py <u>H</u> elp				
Number	Description	Status	Time	ProcessID	
1	<pre>IviScope_init ("SampleScope",VI_TRUE,VI_TRUE,0x00D5251C)</pre>	0	17:54:04.410	0xFFFD5435	
2	IviScope_ConfigureAcquisitionType (0x004659C0,0 (0x0))	0	17:54:06.000	0xFFFD5435	
3	lviScope_ConfigureChannel (0x004659C0,"CHANNEL1",5.000000 (5.000000E+000),0.	0 0	17:54:06.000	0xFFFD5435	
4	IviScope_ConfigureChanCharacteristics (0x004659C0, "CHANNEL1", 1000000.0000000)	1 0	17:54:06.000	0xFFFD5435	
5	IviScope_ConfigureAcquisitionRecord (0x004659C0,0.002000 (2.000000E-003),1000 (0	Ix 0	17:54:06.000	0xFFFD5435	
6	lviScope_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 (0.00000	0 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, p	press F1	Captured: 10	, Displayed: 10		

NI Spy displays the name of each class driver function call that it captures. For each function call, NI Spy displays the values of the input and output parameters. You can configure NI Spy to display the following details about 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 it captures. To see the detailed information for a specific call, use one of the following methods:

- 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 click **Properties**.

The Property Sheet dialog box contains additional information when it shows a call to an IVI class driver. Refer to *NI Spy Help* for more information about the NI Spy Property Sheet dialog box.

Input Tab

The following figure shows the Input tab of the Property Sheet dialog box.

II Spy Property Sheet: IviScope_ConfigureChannel ? 🗙				
		[
Input Parameter	Value			
vi channel range offset coupling probeAttenuation enabled	SampleScope (0x00438C40) "CHANNEL1" 5.000000 (5.000000E+000) 0.000000 (0.000000E+000) 1 (0x1) 1.000000 (1.000000E+000) VI_TRUE			
Help on <u>Attribute</u>				
8	Previous <u>N</u> ext	Close		

The Input tab displays the input parameters value for a function call. When the class driver initialize function is called, it passes 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** input parameter and the logical name that corresponds to the value. Thus, you can easily identify the particular instrument session that a call accesses.

Output Tab

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.

The following figure shows the Output tab of the NI Spy Property Sheet with information from a call to an IVI class driver.

II Spy Property Sheet: IviScope_ReadWaveform 🛛 🔋 🗙				
General Input Output Buffer Interchange Warnings				
Output Parameter	Value			
*actualPoints 1000 (0x3E8) *initialX 0.000000 (1.320000E-008) *xIncrement 0.001000 (1.000000E-003)				
Function returned: 0 (VI_SUCCESS)				
	Previous <u>N</u> ext	Close		

Interchange Warnings Tab

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. The following figure shows the Interchange Warnings tab.

NI Spy Property Sheet: IviScope_ReadWaveform	? ×
General Input Qutput Buffer Interchange Warnings	
_ Interchangeability Warnings	
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	•
Previous Next Clo	se

Coercions Tab

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. The following figure shows the Coercions tab.

NI Spy Property Sheet: IviScope_Se	etAttrib	uteViRe	eal64	? ×
General Input Output Coercions				
Au Tute Considera				
Attribute Loercions				
Attribute	Ch	Value	Coerced	ITo
TKDS30XX_ATTR_HORZ_TIME		0.800	1.000	
•				
Provinue	1	Mout	1	
		Меж		Liose

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 Coercions tab also displays the channel name.

Verifying Instrument Replacement Candidates

Click the following links to view techniques that you can use to verify whether a new instrument can work with an existing application.

Developing a Reference Program

Testing the New Driver in Simulation Mode

Developing a Reference Program

One technique for verifying instrument replacement candidates is to create a reference program. You can 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 programmatically describe all the instrument configurations and operations that you require.

To verify whether a new instrument can work with your program, execute the reference program using the specific driver for the new instrument. You can run the reference program with the physical instrument or with <u>simulation enabled</u>. If you can execute the reference program without errors using the new instrument, then you can probably 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 you cannot make connections to the physical instrument, you can <u>enable</u> <u>simulation</u>.

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 you can probably replace your existing instrument with the new one.

IVI Class Driver Simulation Overview

The IVI class drivers implement simulation features by using simulation drivers. ICP 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.

The following table lists the simulation driver files that ICP installs.

IVI Class Simulation Drivers

Class Drivers	Simulation Driver Files
IviScope	nisScope.dll, nisScope.c, nisScope.h, nisScopu.uir, nisScopeUir.c, nisScopeUir.h
lviDmm	nisDmm.dll, nisDmm.c, nisDmm.h, nisDmmu.uir, nisDmmUir.c, nisDmmUir.h
lviFgen	nisFgen.dll, nisFgen.c, nisFgen.h
IviSwtch	nisSwtch.dll, nisSwtch.c, nisSwtch.h
lviDCPwr	nisDCPwr.dll, nisDCPwr.c, nisDCPwr.h, nisDCPwru.uir, nisDCPwrUir.c, nisDCPwrUir.h
Common Files	nisimu.uir, nisGeneralUir.c, nisGeneralUir.h

ICP 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 ICP installation program places the simulation driver .dll files in the *<IVI Standard Root Dir >*bin directory. The installation program places the .c, .h, and .uir files in the *<All Users>\Documents\National Instruments\CVI\Samples\IVI\ClassSim directory.*

Configuring Simulation

Refer to **National Instruments IVI Driver Help**»**Configuring Your System** for more information about configuring simulation in MAX.

Configuring the Simulation Driver Session

Refer to **National Instruments IVI Driver Help»Configuring Your System** for more information about configuring the simulation driver session in MAX.

Interactive Simulation

In all class drivers, the VXI*plug&play*-compliant VIs and functions have output parameters that return values or strings from the instrument.

VXI <i>plug&play</i> -compliant VIs	VXIplug&play-compliant Functions
<u>ClassPrefix Self-Test</u>	 <u>CLASSPREFIX_self-test</u>
<u>ClassPrefix Revision Query</u>	 <u>CLASSPREFIX_revision_query</u>
<u>ClassPrefix Error-Query</u>	 <u>CLASSPREFIX_error_query</u>

During simulation, the simulation driver can create simulated data for the output parameters of these functions. This panel allows you to configure the simulation data to return from these functions.

You can access this view in the IviDmm, the IviDCPwr, or the IviScope Simulator Setup dialog boxes. In this case, you see the Simulator Setup dialog box for IviScope. Scroll down to see a description of each control.

虁 lviScope Si	nulator Setup	×
View:	XI Plug&Play Functions Simulation	Simulation Driver
Self-Te Fail Self-Te	t Function Pass the results you want the se Self-Test Code: st Self-Test Message: No Error	If-test function to return.
Errory- ERRY ERRY Error	Query Function Pass the results you want the er Error Code: <u>*</u> Error Message: Simulating	ror-query function to return.
Revisio Revisi	n Query Function Pass the results you want the re Instrument Driver Revision: Simulating Firmware Revision:	vision-query function to return.
Help		OK Cancel

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 MAX to specify attribute values in the Initial Settings tab of the Simulation Driver Sessions folder. If you do specify initial settings, however, the values that you specify appear as the initial values in the Simulator Setup dialog box each time you call *ClassPrefix_*init.

Self-Test

Use the Result Code and Message controls to specify the values you want the ClassPrefix Self-Test VI or *ClassPrefix_*self_test function of the driver to return in its output parameters.

Error-Query

Use the Error Code and Message controls to specify the values you want the ClassPrefix Error-Query VI or *ClassPrefix_*error_query function of the driver to return in its output parameters.

Revision Query

Use the Instrument Driver Revision and Firmware Revision controls to specify the values you want the ClassPrefix Revision Query VI or *ClassPrefix_*revision_query function of the driver to return in its output parameters.

VXIplug&play Function Simulation

To configure the simulation of the VXI*plug&play* functions, select **VXIplug&play Function Simulation** from the View ring control of the Simulator Setup dialog box. 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.

Controls in the VXI*plug&play* Function Simulation View

Control Label	Description	Related Attribute in the Initial Settings 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

You use the dialog box shown below to configure the status code for a simulation. You can access this view in the IviDmm, the IviDCPwr, or the IviScope Simulator Setup dialog boxes. In this case, you see the Simulator Setup dialog box for IviScope.

To configure the status code simulation, select **Status Code Simulation** from the View ring control.

Scroll down to see a description of each control.

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

This control lists all of the class driver functions that the selected instrument specific driver supports. The list also indicates each function's associated simulation status code in both macro name form and in hexadecimal value form. You use the **Status Code Macro** and **Custom Status Code** controls to modify simulation status codes.

Setting this control has the same effect as configuring the *PREFIX_*ATTR_SIMULATE_STATUS_CODES attribute on the Initial Settings tab of **IVI Drivers**»Advanced»Simulation Driver **Sessions**»nis/viClass in MAX or in the simulation interactive panels.

Status Code Macro

This control allows you to change the simulation status code of the currently selected function. Setting this control to the custom status code option sets the value you placed in the **Custom Status Code** control as the function's simulated status code.

If you select Custom Status Code, you can configure a custom status code in the Custom Status Code control.

Custom Status Code

This control lets you specify a custom status code to simulate for a particular function.

• Reset all to VI_SUCCESS

This control resets all of the class driver functions' simulation status codes to VI_SUCCESS (0).

Configuring simulated status codes with this dialog box has the same effect as configuring the corresponding simulated status code attributes on the Initial Settings tab of **IVI Drivers»Advanced»Simulation Driver Sessions»nis/viClass** in MAX or in the simulation interactive panels.

Related Topics

For controlling simulation in LabWindows/CVI, refer to:

Attributes for Controlling IviDmm	Attributes for Controlling IviScope
Simulation	Simulation
Attributes for Controlling IviFgen	Attributes for Controlling IviSwtch
Simulation	Simulation
Attributes for Controlling IviDCPwr Simulation	
For controlling simulation in Lah	/IEN/ rofor to:
i or controlling sinitiation in East	TEVV, Telef to:
Properties for Controlling IviDmm	Properties for Controlling IviScope
Simulation	Simulation
Properties for Controlling IviDmm	Properties for Controlling IviScope
Simulation	Simulation
Properties for Controlling IviFgen	Properties for Controlling IviSwtch
Simulation	Simulation

Properties for Controlling IviDCPwr Simulation

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.

The following figure shows the Measurement Data Simulation view for the IviScope simulation driver.

🐗 lviScope Simulator Setup	×
View: Measurement Data Simulation	Simulation Driver
Always prompt for output data simulation	
Channel: CH1 (Channel1)	
	Waveform: Sine
Signal at the probe tip for the selected channel:	Frequency (Hz):
	Amplitude (Volts): 🚔 10.00000000
	DC Offset (Volts):
	Phase Shift (deg): 🚔 -20.00
	Duty Cycle (%). 🏮 10.00
Time: 4.00000000 milliseconds	SNR Ratio: 🚔 Hnf
Help	OK Cancel

IviScope Measurement Data Simulation View

The Measurement Data Simulation view for each simulation driver has an Always Prompt for Output Data Simulation checkbox control. Use this 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 the data separately that the simulation driver generates for each measurement. When you disable the 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 MAX on the Initial Settings tab of **IVI Drivers»Advanced»Simulation Driver Sessions»nis/viClass** or in the simulation interactive panels.

Non-Interactive Simulation

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

Advanced Simulation Topics

Expand this book for topics about advanced simulation.

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 Compliance Package 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, do 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 require the LabWindows/CVI Run–Time Engine. The IVI Compliance Package 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 LabWindows/CVI 5.0 or later.

LabWindows/CVI 5.0.*x*—To modify and recompile the simulation driver using the LabWindows/CVI 5.0.*x* compiler, complete 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 so that the compiler does not process any of the user interface code.

- Enable the Instrument Driver Support Only command in the Build menu of the Project window. When you enable Instrument Driver Support Only, your project does not link to the LabWindows CVI Run–Time Engine. To enable Instrument Driver Support Only, select Build»Instrument Driver Support Only.
- 3. Select **Build»Create Dynamic Link Library** 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 **Change** 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 **OK** to create the DLL.

LabWindows/CVI 5.5—To modify and recompile the simulation driver using the LabWindows/CVI 5.5 compiler, complete 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 so that the compiler does not process any of the user interface code.

- Select Build»Target Settings in the Project window. The Target Settings dialog appears. Enable Instrument Driver Support Only 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 by completing the following steps:
 - a. Click **Change** in the export section to display the DLL Export Options dialog box.
 - 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.
- 3. Select **Build»Create Release Dynamic Link Library** in the Project window to compile the simulation .dll file.

LabWindows/CVI 6.0 and 7.0—To modify and recompile the simulation driver using the LabWindows/CVI 6.0 or 7.0 compiler, complete 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 so that 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. Select **Instrument Driver Only** from the Runtime Support control. Be sure to export the symbols from the .h file of the simulation driver by completing the following steps:
 - a. Click **Change** in the export section to display the DLL Export Options dialog box.
 - 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.
- 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 LabWindows/CVI Run–Time Engine, but you cannot use the interactive simulation panels. If you attempt to enable interactive simulation without the LabWindows/CVI Run–Time Engine, the simulation driver returns an error.

Enabling Simulation after Initializing

If you have disabled simulation when you initialize the instrument driver, some instrument drivers allow you to 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 topic describes. This approach is useful if you run your application.

IVI Class Driver Help

This book contains technical and programming support for both LabWindows/CVI and/or LabVIEW users for the following National Instruments Interchangeable Virtual Instruments class drivers:

- IviDCPwr
- IviDmm
- IviFgen
- IviPwrMeter
- RFSigGen
- IviScope
- IviSpecAn
- IviSwtch

The book also describes the fundamental and advanced terminology for these IVI class drivers. This book is intended for use by developers with a working knowledge of LabWindows/CVI and/or LabVIEW.

This book is divided into the two following books:

IVI Class Driver Help for LabWindows/CVI

IVI Class Driver Help for LabVIEW

Important Information

Warranty Copyright Trademarks Patents Warning Regarding Use of NI Products

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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about <u>National Instruments trademarks</u>.

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help**»**Patents** in your software, the patents.txt file on your CD, or <u>ni.com/patents</u>.

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.

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- <u>Support</u>—Online technical support resources at ni.com/support include the following:
 - Self-Help Resources—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable <u>KnowledgeBase</u>, <u>product</u> <u>manuals</u>, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - Free Technical Support—All registered users receive free Basic Service, which includes access to hundreds of Applications Engineers worldwide in the <u>NI Discussion</u> <u>Forums</u> at forums.ni.com. National Instruments Applications Engineers make sure every question receives an answer.

For information about other <u>technical support options</u> in your area, visit ni.com/services or <u>contact</u> your local office at ni.com/contact.

- <u>Training and Certification</u>—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- <u>System Integration</u>—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your <u>local office</u> or NI corporate headquarters. You also can visit the <u>Worldwide Offices</u> section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Measurement & Automation Explorer (MAX) Configuration Utility

Use Measurement & Automation Explorer (MAX) configuration utility to configure your instrument-independent test system. With MAX, 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 (also known as: hardware assets) and specific drivers (also known as: software modules) to use.



Note MAX is not included in the IVI Compliance Package available for download from ni.com/idnet.

Branch Offices

Office	Telephone Number
Australia	1800 300 800
Austria	43 0 662 45 79 90 0
Belgium	32 0 2 757 00 20
Brazil	55 11 3262 3599
Canada	800 433 3488
China	86 21 6555 7838
Czech Republic	420 224 235 774
Denmark	45 45 76 26 00
Finland	385 0 9 725 725 11
France	33 0 1 48 14 24 24
Germany	49 0 89 741 31 30
India	91 80 41190000
Israel	972 0 3 6393737
Italy	39 02 413091
Japan	81 3 5472 2970
Korea	82 02 3451 3400
Lebanon	961 0 1 33 28 28
Malaysia	1800 887710
Mexico	01 800 010 0793
Netherlands	31 0 348 433 466
New Zealand	0800 553 322
Norway	47 0 66 90 76 60
Poland	48 22 3390150
Portugal	351 210 311 210
Russia	7 095 783 68 51
Singapore	1800 226 5886
Slovenia	386 3 425 4200

South Africa	27 0 11 805 8197
Spain	34 91 640 0085
Sweden	46 0 8 587 895 00
Switzerland	41 56 200 51 51
Taiwan	886 02 2377 2222
Thailand	662 278 6777
United Kingdom	44 0 1635 523545
United States (Corporate)	512 683 0100