Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters
11500 North Mopac Expressway  Austin, Texas 78759-3504  USA  Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, 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 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the Additional Information and Resources appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.
## Contents

### Student Guide
- A. NI Certification ........................................................................................................... v
- B. Course Description ......................................................................................................... vi
- C. What You Need to Get Started .................................................................................... vi
- D. Installing the Course Software ...................................................................................... vi
- E. Course Goals .................................................................................................................. vii
- F. Course Conventions ....................................................................................................... viii

### Lesson 1
#### Industry Applications
- A. Introduction to Instrument Control ........................................................................... 1-2
- B. Benefits of Instrument Control ................................................................................... 1-3

### Lesson 2
#### Communicating with Instruments on Different Buses
- A. Introduction to Instrument Control Buses ................................................................... 2-3
- B. Connecting to Instruments .......................................................................................... 2-4
- C. Monitor Communications ............................................................................................. 2-32
- D. VISA ............................................................................................................................ 2-34
- E. Programming with SCPI ............................................................................................... 2-56
- F. Parsing Data .................................................................................................................. 2-59
- G. Instrument I/O Assistant .............................................................................................. 2-66

### Lesson 3
#### Using Instrument Drivers
- A. What is an Instrument Driver ....................................................................................... 3-2
- B. When to Use an Instrument Driver ............................................................................. 3-4
- C. Installing an Instrument Driver .................................................................................... 3-8
- D. Instrument Driver VIs ................................................................................................... 3-16
- E. Controlling an Instrument ............................................................................................. 3-19

### Lesson 4
#### Modifying an Existing Instrument Driver
- A. When to Modify an Instrument Driver ........................................................................ 4-2
- B. Instrument Driver Internal Design Model .................................................................... 4-6
- C. Steps to Modify an Existing Instrument Driver ............................................................ 4-10
Contents

Lesson 5
Creating a New Instrument Driver
A. LabVIEW Instrument Driver Development Process ............................................5-4
B. Designing Your Instrument Driver API.................................................................5-7
C. Testing an Instrument Driver ..............................................................................5-81
D. Deploying an Instrument Driver ...........................................................................5-96

Lesson 6
Developing an Application
A. Instrument VI Architectures .................................................................................6-2
B. Guidelines .............................................................................................................6-13

Appendix A
Additional Information and Resources

Course Evaluation
Lesson 5: Creating a New Instrument Driver

Sometimes you have an instrument that does not have an instrument driver. This lesson describes how to create a new instrument driver. Topics include:

• Designing an instrument driver structure
• Developing an instrument driver
• Testing an instrument driver
• Deploying an instrument driver
Due to the wide array of instruments and instrument control interfaces, it is important to choose the proper method of instrument control. The following flowchart can help you choose the appropriate method of instrument control.

If an instrument driver is not available, you can create a new instrument driver. Before creating a driver, consider searching for similar drivers and using an existing driver as the starting point for the new driver. Even if you cannot find a similar driver, you can still reduce development time by using pre-defined driver templates.
Reasons to Create a New Instrument Driver

• An instrument driver does not exist for your instrument
  – You need to send many commands to your instrument
  – You need to distribute the created instrument driver to other developers
  – You can use the created instrument driver communicate with your instrument in future applications
• An existing instrument driver lacks many functions that you need and requires several modifications

If you need an instrument driver for your instrument, but no instrument drivers exist or existing instrument drivers require several modifications, then you may want to create a new instrument driver.
A. LabVIEW Instrument Driver Development Process

1. Design
2. Develop
3. Test
4. Deploy

ni.com/training
Before You Develop Your Driver

- Familiarize yourself with the instrument
  - Read the operating manual
  - Use the instrument manually and in an application
  - Set up a simple test
- Familiarize yourself with the programming manual
  - Read information on the instruction set
  - Determine which features you want to access
  - Test sending commands

API Design
The user interacts with the instrument driver through the Application Programmer Interface (API). API design is the most important aspect of Instrument Driver development, but often gets the least attention. You should start by familiarizing yourself with the instrument. This preparation is necessary so that the instrument’s command set can be organized into an intuitive set of VIs.

Familiarize yourself with the instrument, its capabilities, and use
Being familiar with the instrument and its uses in actual applications helps you design a useful instrument driver API. Read the operating manual thoroughly. Learn how to use the instrument manually before you attempt developing interfaces to it. Use the instrument in an actual application to get practical experience. If the operating manual explains how to set up a simple test, do so.
Familiarize yourself with the instrument programming manual.

Study the programming section of the instrument manual. Read the information on the instruction set to familiarize yourself with the available controls and functions and the organization of the features. Determine which instrument features you want to access for programmatic use. Test sending commands to the instrument to understand how the commands affect the operation of the instrument. Use one of the following tools to aid in interactive testing.

**Instrument I/O Assistant**—The Instrument I/O Assistant Express VI, available with LabVIEW 7 Express or later in the Express»Input or Express»Output palette, lets you send commands to an instrument, then view and parse the responses in real-time using steps. When you place the assistant on the block diagram, the interactive window appears. Though the assistant provides simple instrument control when an instrument driver is unavailable, you can use it as an interactive tool as well.

**VISA Interactive Control**—NI-VISA includes the VISA Interactive Control (VISAIC) utility, which is called on all platforms that support the LabWindows/CVI Run-Time Engine. This utility provides access to all VISA functionality interactively in a graphical environment. Use this utility to test instrument commands as you develop instrument drivers.

**Communicate with Instrument Utility**—(Windows Only) Select the Devices and Interfaces option in MAX to communicate with IEEE-488.2 instruments. Use this utility to read, write, and query an instrument in a graphical environment. Use this utility to test instrument commands as you develop instrument drivers.
B. Designing Your Instrument Driver API

You can use existing certified instrument drivers or the LabVIEW instrument templates as a reference of an API for a similar instrument.

- Start with templates or a similar existing certified driver
- Use good API practices
  - Intuitive grouping of instrument commands per VI
  - Consistent with itself and similar APIs
  - Usable for simple and advanced applications
An instrument perspective of an Instrument Driver API is displayed in its command set.

It is important for the developer to group appropriate instrument functionality when planning the instrument driver API. In this example, waveform function, amplitude, DC offset, frequency, and duty cycle are grouped as this functionality is commonly considered at the same time by the user. Another example of grouping common functionality would be including Trigger Source, Trigger Level, and Trigger Slope parameters for a trigger VI. Effective grouping of instrument functionality in a driver can help simplify things for a user. A driver should never be created with one VI per operation in a get/set manner because this adds no value to the API.

The developer should reference the instrument API presented in the command set, but deviate when value can be added. LabVIEW instrument driver VIs present the low level instrument command set in higher level, easy to use VIs.
A LabVIEW perspective of an Instrument Driver API is displayed in the VI Tree.
The ke27xx instrument driver’s API is displayed in its VI Tree. The Initialize, Close, Reset, Self-Test, Revision Query, Error Message, and Error Query are all required functions for LabVIEW Plug and Play instrument drivers. High level application examples should also be provided. Application examples can help the user get started quickly and serve as a teaching tool for using the instrument driver.
Lesson 5  Creating a New Instrument Driver

Exercise 5-1: Get to Know the Instrument

Familiarize yourself with the NI Oscilloscope Simulator and its programming manual.
After designing your instrument driver API, you are ready to develop the instrument driver.
Adhere to design guidelines and be consistent when developing the instrument driver API. Doing so ensures consistency within the API itself and with the API for drivers for similar instruments. You should use good API practices while designing your API.

Develop your instrument driver so it is consistent with the other VIs in your API and consistent with other instrument drivers that control similar instruments. Developing VIs that deviate from established conventions can detract from the user experience, even if your design improves usability, because users must relearn a user interface.

Follow these guidelines to create an instrument driver that provides a consistent API.

The following list includes several other API tips from the *Advanced LabVIEW Design–Usability, Reusability, and Maintainability* session.

**Front Panel / Connector Pane**

Use consistent Front Panel & Connector Pane layout

**Icons**

Should have unifying theme and use meaningful glyphs
Datatypes
Be consistent:
Use int32s and float64s for most numbers
Makes future API changes easier
Example: with enum of “handshaking modes”, we might add new modes in the future.
Make clear the sense of Booleans:
Unclear: “Polarity (T)”
Clear: “Positive Polarity (T)”
Avoid reverse logic Booleans
Confusing: “Don’t Convert EOL (F)”
Better: “Convert EOL (T)”

Naming
Use self-documenting, descriptive parameter and VI names
VI Names
Use a consistent prefix (for example, “ke27xx”) for all VI names, including subVIs that aren’t part of the API
The Instrument Driver Guidelines explain the National Instruments requirements and recommendations for creating a certified LabVIEW Plug & Play instrument driver. Use the guidelines to review your driver for instrument driver certification. Because end users look for certification as an indicator of a high quality instrument driver, National Instruments reviews submitted instrument drivers against these requirements and recommendations.

The Instrument Driver Guidelines include the following 12 main sections:

- Before You Develop Your Driver
- Driver Architecture and API Design
- VIs: Names and Properties
- Controls/Indicators: Naming and Data Representation
- VI Front Panels
- Icon and Connector Panes
- Block Diagrams
- Testing
- Documentation
- Example VIs
- Palette Menu Files
- Files and Documents to be Submitted
Use a consistent front panel and connector pane layout. The top inputs and output are most commonly used for passing the VISA resource name and the bottom inputs and outputs are used for error handling.
What is wrong with this VI?

Guidelines

Avoid using strings as controls and indicators

VISA resource name
error in (no error)

VISA resource name out
Voltage
error out

Bad Design
The Voltage indicator wire is pink, which indicates that it is a string data type. Because Voltage represents a numeric value, you must convert it from a string data type to a numeric data type in order to use it in most cases. The corrected Voltage data type is orange, which is the double precision numeric data type, which is much more useful.
Guidelines

VI and Control Naming and Help

Mode.vi

VISA resource name

Value

error in (no error)

Sets the Mode value.

Bad Design

This function in the programming manual might be called “mode”, but we don’t want the driver user to have to use the programming manual in order to understand what this function does.
We can improve this VI by adding descriptive naming and more intuitive data types.

Improvements:
• More descriptive VI name
• More descriptive input name than “Value”
• “Value” input was changed from a double datatype to a Boolean since it is an “Enable” control that only has two states.
Your driver might be distributed internationally and it is recommended that you choose icons that are engineering focused instead of relying on terms that might only make sense in English, such as trigger.
Appropriate icon choice will help make your instrument driver more usable for both domestic and international users. They also increase the overall impression of your driver.
The Icon Art Glossary includes graphics to help you create LabVIEW VI Icons faster. Using meaningful graphics also helps identify the operation of your VIs and gives your VIs a more professional look. Each graphic corresponds to an operation, such as aborting an I/O session or controlling the mode of an instrument. You also can use these graphics for other things that might require glyphs, such as in user interface graphics and icons for executables.

The Icon Art Glossary also includes a link to the Guidelines for Creating Icons.

You can access the Icon Art Glossary by clicking on the Icon Art Glossary button in the Icon Editor. This will link you to the Icon Art Glossary on ni.com.

Note: Open the Icon Editor by right-clicking the Icon/Connector Pane of your VI and selecting Edit Icon.
Design VIs that give users the appropriate amount of control of their instruments. Design VIs for reuse as subVIs.

An instrument driver with a few high-level VIs might not give the user enough control of the instrument operation. An instrument driver with many low-level VIs might be too complicated for users unfamiliar with instrument rules regarding command order and interaction. Understanding the application needs of your customer can help you define the correct level of granularity for developing VIs as subVIs. Therefore, when using a measurement device such as an oscilloscope, the user typically configures the instrument once and takes many measurements. In this case, you should write high-level configuration VIs that group dependent or related configuration settings in a single VI. Users typically want to enable a stimulus system or initiate a measurement at a different point in their application. Therefore, VIs that Enable Output or Initiate Acquisition typically have one or no input parameters.

It is acceptable to include low-level VIs that allow users more control over instrument operations. However, if included, they should not cause confusion to the majority of users. Place low-level VIs several layers down the VI hierarchy. Generally, include a high-level VI that incorporates several similar low-level VIs.
In this slide, the VI from the previous slide is converted into four VIs—Configure Measurement, Configure Aperture, Configure Autozero, and Configure Integration. This offers a better level of granularity and is more useful.
Polymorphic VIs accept different data types for a single input or output terminal. A polymorphic VI is a collection of VIs with the same connector pane patterns. Each VI in the collection is an instance of the polymorphic VI. Build polymorphic VIs when you perform the same operation on different data types.

One example of a good use case for a polymorphic VI is configuration for different types of triggers. The VIs combine into a poly should be mutually exclusive.

Create polymorphic VIs by navigating to **File»New»VI Folder»Polymorphic VI**.

**Note** You can build polymorphic VIs only in the LabVIEW Professional Development System.
Demonstration

Polymorphic VI

Create a polymorphic VI
The internal structure of the instrument driver defines the organization of the instrument driver VIs. All user-accessible API VIs are organized into a modular hierarchy based on instrument functionality.

The figure shows the internal structure of LabVIEW instrument drivers. End users have the necessary granularity for controlling instruments properly in applications. For example, end users can initialize all instruments once at the start, configure multiple instruments, and then trigger several instruments simultaneously. Also, they can initialize and configure an instrument once, and then trigger and read from the instrument several times.

For end users, the model provides a consistent instrument driver model. After users understand one instrument driver, they can apply that knowledge to other LabVIEW instrument drivers.
The Instrument Driver Wizard greatly reduces the time needed to create LabVIEW Plug and Play instrument drivers. The new wizard supports the most common instrument types and walks you through the process of selecting the instrument type, naming the driver which will be used throughout, and creating a unifying VI icon banner. The Wizard also installs the driver automatically so that a user will see the new driver functions in the palette under “Instrument Drivers.” Upon completion, the wizard presents you with a fully documented instrument driver based on the industry-defined Standard Commands for Programmable Interfaces (SCPI) protocol. Furthermore, the generated documentation guides you through the process of modifying the source code to support your instrument-specific commands.
The Instrument Driver Wizard begins by giving the user a firm starting point for creation. The graphic shows the beginning screen for the Instrument Driver Wizard. Here is where you have options to select the foundation for your driver. You can select an existing driver or one of the templates that correspond to many classes of instruments. This step is very important, because it will shape the way the driver looks, feels, and how it fits into the guidelines for instrument driver creation.
First you must create a custom banner for your VIs. This banner ideally refers to the instrument you are working with. It will be appended to all of the icons created for this driver. For example, if your instrument is called the “XYZ” you should put these letters into the banner, so that all VIs for this instrument will show XYZ at the top.
Second, you will want to customize the Top Level Icon to be unique from others. This will be the icon that shows up in the functions palette in LabVIEW. You will want the user to be able to quickly scan all instrument drivers on the computer and find the correct one from the icon, the same way one might scan all the math functions to find a particular curve fit.

It will take a few minutes to finish the two tasks, depending on how creative you decide to get, but once you are done you will have a complete descriptive set of icons with a custom top level. This new feature saves hours of tedious icon work and gives a standard look and feel to your instrument driver that follows all guidelines. Plus, now you have a whole library of icons that can be used later when creating custom instrument-specific VIs.
After you complete the Instrument Driver Wizard, you have spent only a few minutes but have a whole project-style instrument driver. This driver is already ready to run with examples if your instrument follows SCPI protocol. Even if it does not, you have a complete set of VIs with instructions on where to modify in order to become instrument-specific. All of this in just a few easy steps.

The LabVIEW library file type (.lvlib) allows users many advantages. For instance, users can have the same filename in different places with unique LabVIEW Libraries. Resolving these name conflicts is crucial. Think about how many initialize.vi files are on your computer when you download or create many instrument drivers. Furthermore, the LabVIEW Library enables the ability to have public and private VIs. Lastly, LabVIEW libraries can be versioned for organized future modifications.
Lesson 5  Creating a New Instrument Driver

Exercise 5-2: Instrument Driver Wizard

Create a new instrument driver project for the NI Oscilloscope Simulator.
Steps to Create a New Instrument Driver

1. Create an Instrument Driver Project
2. Remove unsupported VIs
3. Modify template VIs
4. Create new VIs
5. Update VI View
6. Update palette manual
7. Update readme

ni.com/training
Removing Unsupported VIs

1. Remove any instrument driver VIs in the project that the instrument does not support, including any common VIs such as Self-Test or Reset. For example, the Digital Multimeter template includes VIs for making multipoint measurements. If you write a driver for a Digital Multimeter that does not support multipoint measurements, delete these VIs from both the driver project and driver directory.

2. For each VI you removed, remove subVI calls to that VI from other instrument driver VIs in the project.

3. Save the project (.lvproj) and project library (.lvlib) after removing files.
Exercise 5-3: Remove Unsupported VIs

Remove unsupported VIs from the instrument driver project.
There are two types of instrument driver VIs that you may need to edit – common driver VIs and instrument-specific VIs.
Modify Common Driver VIs

First, modify the common driver VIs before you modify the instrument-specific VIs. The common driver VIs are the VIs that are generally part of all instrument drivers. Common driver VIs include the Initialize, Close, Self-Test, Reset, Error Query, Revision Query, and Default Instrument Setup VIs.
Modifying Common Driver Steps

1. Read the VI documentation
2. Follow the modification instructions in each VI
3. Delete the instructions
4. Make remaining modifications necessary

First read the VI documentation, follow the modification instructions in each VI, and then delete the instructions. Then, make any remaining modifications necessary.
Modify Common Driver VIs – Initialize

Possible modifications
- Bus configuration
- Identification query response

Initialize
Establishes communication with the instrument and optionally performs an instrument identification query and/or an instrument reset. It also places the instrument in a default state needed for other instrument driver operations. Therefore, call this VI before calling other instrument driver VIs for this instrument. Generally, you need to call the Initialize VI only once at the beginning of an application.

Possible modifications
Each Initialize VI optionally calls an Identification Query that is specific to an instrument model or model series. If you are using a driver built for an instrument model that is similar to but not exactly matching the instrument you use, either turn off this option or change the response to the identification query command. For SCPI instruments, this command is *IDN?.

Read the VI documentation, follow the modification instructions in the VI, and then delete the instructions. Then, follow these additional instructions:
Initialize VI—Switch to the front panel and enter the default address information in the VISA resource name control. Refer to the VISA resource name control context help for the syntax for the device. Right-click the control and select Data Operations»Make Current Value Default from the shortcut menu. Update the context help of this control to show only information about supported buses for the instrument.

Modify or remove serial controls in the Serial Configuration cluster to match the capabilities of the instrument. For example, if an instrument can only have 7 data bits and 1 stop bit, remove the Data Bits and Stop Bits controls and set these parameters as constants on the block diagram. If an instrument only supports 2 baud rates, remove unsupported rates from the Baud Rate control. Set the controls in the Serial Configuration cluster to the instrument defaults, right-click the controls, and select Operate»Make Current Values Default from the shortcut menu.

You also can use the Serial Configuration cluster for other configuration settings such as a non-serial bus like TCP/IP. If necessary, update the cluster name and context help to reflect the controls within the cluster.
Modify Common Driver VIs – Close

Possible modifications
- If necessary, add code to block diagram before the VISA Close function

Close
Performs an instrument error query before terminating the software connection to the instrument.

Modifications
If you need to configure the instrument to a certain state before it can be closed, add the necessary code to the block diagram before the VISA Close function.
Self-test
Runs the instrument’s self-test routine and returns the test results.

Modifications
Modify the self-test command if it is not *TST?.
Modify the return constant for a passed test if it is not “0”.
Set the timeout constant on the block diagram to a value large enough for the instrument to finish its self-test. The default is 20,000 milliseconds, or 20 seconds.
Reset
Resets the instrument and then sends a set of default setup commands to the instrument.

Modifications
Modify the reset command if it is not *RST.
Error Query

Queries the instrument for any errors in the instrument's error queue. It will continue to read errors until all errors have been read from the instrument. Any errors will be returned in the error out control.

Note  This VI is called automatically by most driver VIs so it is not usually necessary to call this in an application.

Modifications

The Error Query VI is designed to parse multiple SCPI errors. If the instrument implements error checking differently, refer to other instrument drivers, such as the Fluke 45 Multimeter, for alternatives to implement this functionality. The Error Query VI is included as a subVI in all driver VIs except low-level API VIs. For SCPI compliant instruments, use \texttt{:SYST:ERR?} for the error query command. For non-SCPI compliant instruments, parse the error query response to see if an error occurred. Based on the response, set the Boolean status element of the error cluster to the appropriate state: The Boolean should be False if there is no instrument error to report, and True otherwise.
Revision Query
Queries the current instrument firmware revision and instrument driver revision.

Modifications
Modify the revision query command if is not *IDN?.
Modify the Scan From String function to parse out the firmware query for the instrument.
Modify Common Driver VIs – Default Instrument Setup

Possible modifications
• Send commands needed to get instrument into desired state

Default Instrument Setup
This Utility VI sends a default command string to the instrument whenever a new VISA session is opened, or the instrument is reset. Use this VI as a subVI for the Initialize and Reset VIs. Altering this VI could cause these two VIs to operate incorrectly.

Modifications
Change the Default Setup String constant to the commands needed to get the instrument into a desired known state for the instrument. For example, you should add a command for setting oscilloscopes to single acquisition mode, such as :ACQ:STOPA SEQ;.
Exercise 5-4: Modify Common Driver VIs

Modify the common driver VIs to work with the NI Oscilloscope Simulator.
Instrument-specific VIs are any VIs in the instrument driver project that are not one of the common driver VIs. This slide show some example instrument-specific VIs. After you are finished modifying the common driver VIs, make any necessary modifications to the instrument-specific VIs.
Modify Instrument-Specific VIs

Possible modifications
- Add, modify, or remove front panel controls and indicators
- Edit block diagram to match instrument functionality
- Update the default values of the controls to match the default values of the instrument
- Update labels and descriptions
- If you removed or added controls and indicators, edit the connector pane to match

Modifying Instrument-Specific VIs
Complete the following steps for each remaining VI in the driver.

1. Open the VI and edit the front panel by adding, modifying, or removing controls and indicators as needed to perform the operation defined by the VI.

2. Update the default values of the controls to match the default values of the instrument. Set the value on each control and then right-click the control and select Data Operations » Make Current Value Default from the shortcut menu.

3. Update the control and indicator labels and descriptions to reflect any changes.

4. Edit the VI icon as needed.

5. If you removed or added controls and indicators, edit the connector pane to match.

6. Edit the block diagram by updating the command strings and parsed responses to match instrument functionality.

7. Edit the VI documentation as needed. An example of documentation you could enter for the Initialize VI is: This instrument driver supports serial communication. If you encounter problems when using serial communication, try using baud rates less than 19,200 and hardware handshaking.

8. Save and test the VI.

9. Save the project (.lvproj) and project library (.lvlib) after modifying files.
Exercise 5-5: Modify an Instrument-Specific VI

Modify the instrument-specific Fetch Waveform VI to work with the NI Oscilloscope Simulator.
After you are finished modifying the necessary VIs in your instrument driver, you might need to create additional VIs for the instrument driver. There are two types of instrument driver VIs you can create—instrument-specific VIs and example VIs.
If you need to add additional functionality to your instrument driver, then create new instrument-specific VIs for your instrument driver.
Instrument Driver VI Wizard

- Create additional VIs for your instrument
- Inherits Instrument Driver Properties
  - Icon banner
  - Library
- Generates LabVIEW code

**Instrument Driver VI Wizard**

Use the Instrument Driver VI Wizard to create additional VIs that the instrument supports. You should create additional VIs for features you need that are currently not included in your instrument driver project. The Instrument Driver VI Wizard allows you to configure and generate VIs that inherit the instrument driver icon banner and library.
VI Creation Options

- Create from an instrument driver template VI
- Copy from an existing VI
- Configure your own VI
- Add a blank VI

VI Creation Options

- Create from an instrument driver template VI—Creates the instrument driver VI from a template VI.
- Copy from an existing VI—Copies and configures an existing instrument driver VI and its subVIs.
- Configure your own VI—Configures an instrument command, specifies how to parse a response, interacts with connected instruments, and generates the corresponding instrument I/O and string formatting/parsing code into a new instrument driver VI.
- Add a blank VI—Creates an instrument driver VI that contains only a VISA resource name control and indicator and an error cluster control and indicator.
Demonstration

Instrument Driver VI Wizard

Create new instrument-specific VIs using the Instrument Driver VI Wizard.
Exercise 5-6: Instrument Driver VI Wizard

Create an instrument-specific VI to configure the NI Oscilloscope’s sample count.
After you are done creating your instrument-specific VIs, create example VIs for your instrument driver. Example VIs show the user the recommended way of using your instrument driver.
Examples should help users understand the recommended calling sequence necessary to perform instrument operations. Include an example for each typical use case for the instrument to benefit a wide range of users. If you modified an instrument driver to add new functionality, add an example VI to the instrument driver project to demonstrate the new functionality.
Creating Example VIs

1. Create two or more example VIs that demonstrate how to use the instrument driver. Examples should help users understand the recommended calling sequence necessary to perform instrument operations. Include an example for each typical use case for the instrument to benefit a wide range of users.

2. Include example VIs in the instrument driver project, but not in the project library.

3. Select **Tools»Prepare Example VIs for NI Example Finder** to add an example data file (.bin3) to the examples directory in the instrument driver project.
Create Example VIs

- Show users how to build their applications
- Self-contained and runnable
- Do not wire terminals to connector pane
- Create separate example for each typical use case
- Give the example a meaningful name

Example VI Guidelines

1. Create example VIs that show users how to build their applications.
   Provide examples with source code. Call user-accessible VIs to teach users how to control the instrument. Examples help users understand the recommended calling sequence necessary to perform instrument operations. If a simple example calls an excessive number of API VIs, the instrument driver might need a more appropriate level of granularity.

2. Create self-contained and runnable example VIs.
   Examples are self-contained and runnable VIs that perform complete operations. For example, a good example VI for a DMM configures, starts, and reads from the DMM. Include calls to the Initialize, Close and Simple Error Handler in your examples. Also, example VIs help users establish communication with the instrument and learn how to programmatically control the instrument.
3. Do not wire terminals to the connector pane of an example VI. Do not design example VIs to be used as subVIs.
   Example VIs are not part of the instrument driver API and, therefore, users should not use them as subVIs. Examples might change from one driver release to the next. If you wire the terminals to the connector pane, end users might use the example VI in their applications. A change to the example in a future release would break their application.

4. Create a separate example for each typical instrument use case.
   Users expect most instrument drivers to include an example for each typical use case for an instrument. Including multiple examples benefits a wider range of customers.

5. Give the example a meaningful name.
   Use names in the example VIs that describe the operation the VI performs, such as “AG33xxx Generate Arbitrary Waveform” or “TKTDS3K Acquire Continuous Waveform”.
Include Example VIs in Project

Include example VIs in the instrument driver project, but not in the project library.

Notice in the project that the Examples folder is not contained inside the project library (.lvlib file).
Prepare Example VIs

Select Tools»Prepare Example VIs for NI Example Finder to add an example data file (.bin3) to the examples directory in the instrument driver project.
Prepare Example VIs

Document Examples page

Use this dialog box to prepare example VIs you create to appear in the NI Example Finder.

Preparing Example VIs to Appear in the NI Example Finder

Complete the following steps to prepare example VIs you create to appear in the NI Example Finder and build the data (.bin3) file that stores the task-based browse paths, keywords, and other information about the example VIs you document. LabVIEW uses the .bin3 file to display example VIs in the NI Example Finder.

1. Select **Tools»Prepare Example VIs for NI Example Finder** to display the Prepare Example VIs for NI Example Finder dialog box.

2. Click the browse button next to the Start directory for examples text box, navigate to the directory that contains the example VIs or projects you want to appear in the NI Example Finder, and click the Current Folder button.

3. In the double-click the example to document list, navigate to and double-click the example VI you want to document by adding task-based browse paths, keywords, and a description. The VI name appears in the Documentation table. The ADE (application development environment) and version of the ADE in which you created the example VI or project appear in the Documentation table when you scroll up. You cannot change the ADE and Version values.
4. Add task-based browse paths to the example VI for users to navigate to the example VI by task on the Browse page of the NI Example Finder.
   a. In the double-click to add a browse path list, choose an appropriate task-based browse path for the example VI.
   b. Navigate to the lowest level of the category, then double-click the gray icon to add the category.

The task-based browse path appears in the Documentation table. You also can copy browse paths from one example VI to another. Double-click the browse path in the Documentation table to remove the browse path from the example VI.

5. (Optional) Repeat step 4 to add more task-based browse paths to the example VI.

6. Add keywords to the example VI for users to enter on the Search page of the NI Example Finder.
   a. In the Double-click to add a word listbox, scroll through the list to find a keyword and double-click the word to add it to the example VI. The keyword appears in the Documentation table.
   b. Enter a word into the New keyword text box and click the Add New Keyword button to add a new keyword to the Double-click to add a word listbox.
   c. Double-click the new keyword in the Double-click to add a word listbox to add the keyword to the example VI. The keyword appears in the Documentation table.

You also can copy keywords from one example VI to another. Double-click the keyword in the Double-click to add a word listbox to remove the keyword from the example VI. Select the keyword in the Double-click to add a word listbox and click the Remove Keyword button to remove a keyword from the list of keywords.

7. (Optional) Repeat step 6 to add more keywords to the example VI.

   Tip Click the Save Documentation button to save the example VI documentation you have added. After you save the example VI documentation, you can proceed through step 13 or you can close this dialog box and finish documenting the example VI later. Relaunch the Preparing for NI Example Finder dialog box and navigate to the example in the double-click the example to document list to edit the example VI documentation you saved.

8. If you have not entered a description for the example VI, enter a description in the Example description text box. The description appears in the Description text box of the NI Example Finder when a user selects the VI. If you already entered a description for the example VI on the Documentation properties page of the VI Properties dialog box, the description appears in the Example description text box.

9. Click the Save Documentation button to save the documentation to the example VI. Clicking this button overrides any existing task-based browse paths, keywords, and description you entered in the Example description text box or in the example VI on the Documentation properties page of the VI Properties dialog box.

10. (Optional) You also can copy the description from one example VI to another.

11. Repeat steps 1 through 10 for each VI you want to appear in the NI Example Finder.
Prepare Example VIs

Build Data File page

Build Data File Page
Use this page of the Prepare Example VIs for NI Example Finder dialog box to build the data (.bin3) file that stores the task-based browse paths, keywords, and other information about the example VIs you document. LabVIEW uses the .bin3 file to display example VIs in the NI Example Finder. The .bin3 file must reside in a directory within the examples, instr.lib, or user.lib directory.

Build the .bin3 file that stores the task-based browse paths, keywords, and other information about example VIs you document.

Click the Build Data File tab.
1. Click the browse button next to the Data file to create (.bin3) text box, and navigate to the directory that contains the example VI or example VIs you want to appear in the NI Example Finder.
2. Enter a meaningful name for the .bin3 file and include the file extension. For example, enter myexamples.bin3, where myexamples is the name of the .llb that contains the example VIs you created. After you enter a name, click OK.
3. Click the Browse button next to the Directory that contains the examples text box, navigate to the directory that contains the example VIs you want to appear in the NI Example Finder, and click the Current Folder button.
4. Click the Build File button to build the data file.
5. Click the Done button to exit the Prepare Example VIs for NI Example Finder dialog box.
Exercise 5-7: Create an Example VI

Create and add an example VI that demonstrates how to configure the NI Oscilloscope Simulator and acquire waveform data.
Update the VI Tree VI

- Move obsolete VIs
- Modify necessary VIs
- Create new VIs
- Update VI Tree VI
- Update palette menus
- Update readme
Update the VI Tree VI

The VI Tree VI allows end users to view the entire instrument driver hierarchy at once. The VI Tree VI is a nonexecutable VI designed to show the functional structure of the instrument driver.

**Steps to Update the VI Tree VI**

1. Open the VI Tree VI and update the block diagram to reflect changes to the instrument driver. The VI Tree is not a subVI in an application. It is intended as a quick overview of the hierarchy of the driver VIs.
2. Remove any VIs that are not supported by the instrument, including any common VIs such as Self-Test or Reset.
3. Add any public VIs that you added to the driver.
4. Save and close the VI.
Exercise 5-8: Update VI Tree VI

Update the VI Tree VI to reflect the changes you made to the instrument driver.
Update Palette Menus

- Move obsolete VIs
- Modify necessary VIs
- Create new VIs
- Update VI Tree VI
- Update palette menus
- Update readme
Update Palette Menus

- All public instrument driver VIs should be accessible from the palette menu
  - Remove links to removed VIs
  - Add subpalette menus and new VIs
- Palette menu should match layout of VI Tree and project folder hierarchy

Update Palette Menus
After you are done modifying and creating VIs for your instrument driver, you must update the palette menus so that your modified and newly created VIs will be accessible from the functions palette.

1. After all VIs are complete, update the function palette menu files (.mnu) to reflect changes to the instrument driver. All public instrument driver VIs should be accessible from the palette menu. Do not include example VIs in the palette menu.
2. Remove links to any VIs you removed from the driver.
3. Add subpalette menus as needed to enhance the palette usability for new VIs. The palette menu should match the layout of the VI Tree and the project folder hierarchy.
Update Palette Menus

1. After all VIs are complete, update the function palette menu files ( .mnu ) to reflect changes to the instrument driver. All public instrument driver VIs should be accessible from the palette menu. Do not include example VIs in the palette menu.

2. Remove links to any VIs you removed from the driver.

3. Add subpalette menus as needed to enhance the palette usability for new VIs. The palette menu should match the layout of the VI Tree and the project folder hierarchy.
Lesson 5  Creating a New Instrument Driver

Exercise 5-9: Update Palette Menu

Update the palette menus to reflect the changes you made to the instrument driver.
Update Readme File

- Move obsolete VIs
- Modify necessary VIs
- Create new VIs
- Update VI Tree VI
- Update palette menus
- Update readme
Update Readme File

1. Open the Readme.html file in an html editor. The Readme file is an html file that includes a driver overview, requirements for use, supported features, revision history, and other information on using the driver. If you created the driver from a template, the driver includes a Readme.html file with fields that you need to modify denoted in red text. If you created the driver yourself, you need to create the Readme.html file.

2. Update the appropriate fields in the Readme file with information specific to the instrument and instrument driver. If you created the driver from a template, update and replace all instances of red text in the file.


4. Update the Readme file with revision history each time you modify the driver.
Exercise 5-10: Update Readme

Update the readme file to reflect the changes you made to the instrument driver.
LabVIEW Instrument Driver Development Process

Design

Develop

Test

Deploy

ni.com/training
C. Testing an Instrument Driver

- Usability
  - API
  - Examples
- Functionality
  - Bus interfaces
  - Error conditions
  - Corner cases
- Peer review
- See testing section in the Instrument Driver Guidelines

C. Testing an Instrument Driver

An instrument driver is not complete until it is thoroughly tested. Different types of testing can help identify different types of problems. Examples can help to test common functionality, the API, and the overall usability of the driver. The functionality of the driver should be exercised for all bus interfaces (GPIB, serial, Ethernet, USB, and so on). All instrument driver VIs should be run independently with all valid combinations of the parameters. The driver should also be tested for corner cases, advanced use cases, and error conditions.
We will discuss these tests in the following slides.
Functional Tests

- Test each parameter value to ensure that instrument settings are updated appropriately
- Test both API VIs and Example VIs
- Test different VI combinations, calling orders, and dependencies

Test different VI combinations, calling orders, and dependencies. Invalid calling orders or dependencies should be handled robustly by the driver. Test parameters as part of a test framework or test them interactively. Test VIs can be saved and executed on future driver versions. If you test VIs interactively, combine the testing with a final front panel review. This type of testing should be executed throughout development and after the driver is complete.
Error Reporting and Recovery Conditions

Verify that Error Query VI returns appropriate error information. Test that the driver can be run with an instrument that is in an error state.

Test both valid and invalid test conditions. Testing failing conditions validates the error handling mechanisms. Send out of range parameter values, call dependent VIs in the incorrect order, or use other methods to produce instrument errors. Refer to the instrument documentation for a complete list of command, execution, device, and system error messages to test. Test the following conditions to ensure the Error Query VI executes appropriately based on instrument or VISA errors:

- Single Instrument error: Error Query should return the instrument’s error information
- Multiple Instrument errors: Error Query should return a queue of instrument errors
- VISA timeout due to command error: Error Query should return the VISA timeout error
- Non-timeout VISA error: Error Query should return the VISA error

The Initialize VI should clear all instrument errors and place the instrument in a known good state so that the driver can execute properly. Abort an instrument driver example during execution and verify that the example can be run again without error.
There are many possibilities for different behavior when supporting multiple buses. First, the instrument itself might have different behavior for each bus type. Then the actual bus protocol might introduce differences, including time delays. Third, VISA defines different behavior for some functions and properties. Take these differences into account as you develop your instrument driver VIs. Thoroughly test the driver with each interface. The majority of testing can be executed on one interface bus. The most common use cases, like the instrument driver examples, should be tested with various parameter selections on all supported buses. Test driver VIs that transfer large amounts of data to or from the instrument. Verify the timeout is appropriate for typical transfers on slow buses. All interface specific code should also be tested.
The instrument driver Readme should include detailed information about any known issues found in the driver. Include information about test coverage for testing performed on the driver. Instrument functionality not supported by the driver should also be documented in the Readme. All of this information can be useful to users of the driver, particularly if problems are encountered.
Timing issues can often be difficult to identify and troubleshoot. A majority of timing issues occur with the serial interface or with poorly designed instruments. Developers should use Wait VIs only as required to work around timing issues. This workaround should be documented in the driver code and also in the Readme. This type of documentation can be denoted with a keyword so that a user can search for and identify all instances of this workaround in the driver.
Stress Testing

- Run examples with large channel counts if applicable
- Test communicating with channels in non-sequential order
- Test different use cases of looping sections of code
- Run examples for an extended period of time
  - Check for performance degradation and significant increases in memory usage
  - Tools»Profile»Performance and Memory

Run examples with large channel counts, if applicable. For instance, if the instrument supports more than one channel, test all channels for different types of operations. Test reading from or writing to channels in non-sequential order.

Run examples for an extended period of time. Verify that there is no degradation in performance and that the driver continues to function as expected. Verify that there is no significant increase in memory usage.

Test different use cases of looping sections of code in a typical series of instrument driver VI calls including entire series of VI calls, the measurement section of code, and multiple configure and measurement sequences.

You can use the Profile Performance and Memory window to acquire and display data about the execution time and memory usage of VIs. Open this window by navigating to Tools»Profile»Performance and Memory.
Mass compiling simply opens VIs up and closes them in an automated fashion. This has a couple of benefits:

- It checks the existence of the subVIs and relinks the subVIs to the main VI. This speeds the loading of VIs because LabVIEW does not have to search for VIs.
- It updates VIs to that version of LabVIEW.
- It reports VIs that have corruption which prevents them from loading. If such VIs are found, they can be restored from a backup.

Before submitting the driver for certification, mass compile the driver to ensure that the instrument driver VI library contains only runnable VIs. In addition to mass compiling the library, review all the VIs in the VI Library. Sometimes temporary or testing VIs appear in the VI Library unintentionally. Mass compile the VIs on a separate computer running LabVIEW to ensure that there are no dependencies to other VIs not installed on the end-user computers. Do so especially if you create a new instrument driver from an existing instrument driver.
Validate that default values in each VI are appropriate for the most common use cases. Instruments often require stimulus to perform measurements. Set examples to use instrument on-board test signals if possible

- Probe compensation signal on an oscilloscope

- If no on-board test signals are available, clearly document required setup and instructions on front panel

Out-of-the-Box Experience

Test example VIs for the “out-of-the-box” experience

- Default values appropriate for common use cases
- Use instrument on-board test signals if possible

Validate that default values in each VI are appropriate for the most common use cases. Instruments often require stimulus to perform measurements. Set examples to use instrument on-board test signals, such as the probe compensation signal available on most oscilloscopes. This provides users with a good out-of-the-box experience when using both the instrument and instrument driver. If no on-board test signals are available, clearly document the required system setup and stimulus needed to perform the operation. Include these instructions on the front panel of the instrument VI.
Firmware and Hardware

- Test driver with the latest firmware
  - Document firmware version tested in Readme
- Test driver on different machines

Firmware and Hardware
Test the driver with the latest instrument firmware and on different machines. Report the instrument firmware version tested in the Readme.
Exercise 5-11: Test a VI

Test valid and invalid inputs to the Configure Sample Count VI.

GOAL

ni.com/training
The National Instruments LabVIEW VI Analyzer Toolkit includes 58 tests that you can run on any VI to easily detect potential problems. The toolkit saves valuable time by quickly identifying ways to improve your VI style, performance, and documentation, and by providing documentation for making changes.

The tests look at everything from the user interface to memory usage and make recommendations for improving your VIs. The toolkit interactively provides direct links to the parts of the VI identified to save you time while editing your VIs. When you complete your testing, use the LabVIEW VI Analyzer Toolkit to create reports in a variety of formats. These reports provide a framework for developing software applications that are flexible, robust, and optimized for performance.

The VI Analyzer toolkit has been customized to test instrument drivers against the Instrument Driver Guidelines. Several tests have been created for instrument driver to supplement the toolkit. Also, a custom configuration file is provided to appropriate priorities for each test.

The VI Analyzer Toolkit and results from testing a specific instrument driver will be demonstrated. Contact instrument.drivers@ni.com for more information.
Exercise 5-12: VI Analyzer

Run instrument driver tests on your instrument driver with VI Analyzer.
After you are finished testing your instrument driver, you are now ready to distribute your instrument driver to your end-users.
D. Deploying an Instrument Driver

There are two main ways you can deploy your instrument driver.

1. Distribute your instrument driver by manually sending the instrument driver project files (preferably in a ZIP file) to your end-users.

2. Submit your instrument driver to the instrument driver network. Afterwards, end-users will be able to download and install your instrument driver from the instrument driver network website.
The Instrument Driver Network on National Instruments Web site – ni.com/idnet – is your starting place to download instrument drivers as well as application notes, tools, and guidelines to aid in developing instrument drivers. The Instrument Driver Network is the largest repository of LabVIEW plug and play drivers and has drivers supporting over 5000 instruments from over 225 vendors.
The Instrument Driver Developer Program is a program designed for instrument vendors to develop LabVIEW or LabWindows™/CVI™ instrument drivers. Through this program, instrument vendors and other developers create software instrument drivers that customers use to easily connect to and control their instruments. Because LabVIEW and LabWindows/CVI are leading software development environments in the test, measurement, and automation industry, vendors can quickly provide software solutions for their existing and potential customers.

If you have developed your own instrument driver or have developed a partial instrument driver, consider sharing it with other LabVIEW developers. There are several distribution channels available to you. You can submit instrument drivers to the National Instruments Instrument Driver Network (ni.com/idnet), you can submit partial drivers or instrument control examples to Developer Zone (zone.ni.com), or you can post your instrument driver on your own web site and have NI’s Instrument Driver Network link to your Web site.

**Instrument Driver Developer Program Benefits**

The IDDP also includes several valuable technical and marketing benefits for its members, including opportunities to promote themselves and their products via National Instruments (NI) and the Instrument Driver Network. Some of the benefits include highlights on the Instrument Driver Network homepage (ni.com/idnet); coverage in the Instrumentation Newsletter, an award-winning global measurement and automation publication that is delivered quarterly to almost all National Instruments customers around the world; and instrument driver maintenance for at least one year, which includes updates for new releases of LabVIEW, bug fixes, instrument driver model revisions, and instrument firmware changes.
How to Join the Instrument Driver Developer Program

Joining the Instrument Driver Developer Program is easy. To qualify for inclusion in the program, you must create an instrument driver for your instrument, submit it for review to National Instruments (NI), and then provide your instrument driver for download to your customers. NI will then formally recognize your company as a participating member of the IDDP and will certify your instrument driver if it is maintained on the instrument driver network.
Summary – Quiz

1. The process for creating a new instrument driver involves designing, developing, testing, and deploying your instrument driver.
   a) True
   b) False

Answer is a) True.
Summary – Quiz

2. Which of the following should you do before developing a new instrument driver?
   a) Familiarize yourself with the instrument
   b) Familiarize yourself with the programming manual
   c) Create a polymorphic VI
   d) Create VI Icons

Answer is a & b.
Summary – Quiz

3. Which of the following helps you create a new instrument driver project by helping you select the instrument type, naming the driver, and creating a unifying VI icon banner?
   a) Instrument I/O Assistant
   b) Instrument Driver Wizard
   c) Instrument Driver VI Wizard
   d) Polymorphic VI

Answer is b) Instrument Driver Wizard.
Summary – Quiz

4. Which of the following are valid methods of distributing an instrument driver?
   a) Manually send the files to end-users
   b) Submit your instrument driver to the Instrument Driver Network
   c) Both a & b

Answer is c) Both a & b.
Notes