Instrument Control

This lesson describes instrument control of stand-alone instruments using a GPIB or serial interface. Use LabVIEW to control and acquire data from instruments with the Instrument I/O Assistant, the VISA API, and instrument drivers.

Topics

A. Instrument Control

B. GPIB

C. Serial Port Communication

D. Using Other Interfaces

E. Software Architecture

F. Instrument I/O Assistant

G. VISA

H. Instrument Drivers

A. Instrument Control

When you use a PC to automate a test system, you are not limited to the type of instrument you can control. You can mix and match instruments from various categories. The most common categories of instrument interfaces are GPIB, serial, modular instruments, and PXI modular instruments. Additional types of instruments include image acquisition, motion control, USB, Ethernet, parallel port, NI-CAN, and other devices.

When you use PCs to control instruments, you need to understand properties of the instrument, such as the communication protocols to use. Refer to the instrument documentation for information about the properties of an instrument.

B. GPIB

The ANSI/IEEE Standard 488.1-1987, also known as General Purpose Interface Bus (GPIB), describes a standard interface for communication between instruments and controllers from various vendors. GPIB, or General Purpose Interface Bus, instruments offer test and manufacturing engineers the widest selection of vendors and instruments for general-purpose to specialized vertical market test applications. GPIB instruments are often used as stand-alone benchtop instruments where measurements are taken by hand. You can automate these measurements by using a PC to control the GPIB instruments.

IEEE 488.1 contains information about electrical, mechanical, and functional specifications. The ANSI/IEEE Standard 488.2-1992 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.

GPIB is a digital, 8-bit parallel communication interface with data transfer rates of 1 Mbyte/s and higher, using a three-wire handshake. The bus supports one system controller, usually a computer, and up to 14 additional instruments. The GPIB protocol categorizes devices as controllers, talkers, or listeners to determine which device has active control of the bus. Each device has a unique GPIB primary address between 0 and 30. The Controller defines the communication links, responds to devices that request service, sends GPIB commands, and passes/receives control of the bus. Controllers instruct Talkers to talk and to place data on the GPIB. You can address only one device at a time to talk. The Controller addresses the Listener to listen and to read data from the GPIB. You can address several devices to listen.

Data Transfer Termination

Termination informs listeners that all data has been transferred. You can terminate a GPIB data transfer in the following three ways:

- The GPIB includes an End Or Identify (EOI) hardware line that can be asserted with the last data byte. This is the preferred method.
- Place a specific end-of-string (EOS) character at the end of the data string itself. Some instruments use this method instead of or in addition to the EOI line assertion.
- The listener counts the bytes transferred by handshaking and stops reading when the listener reaches a byte count limit. This method is often used as a default termination method because the transfer stops on the logical OR of EOI, EOS (if used) in conjunction with the byte count. Thus, you typically set the byte count to equal or exceed the expected number of bytes to be read.

Data Transfer Rate

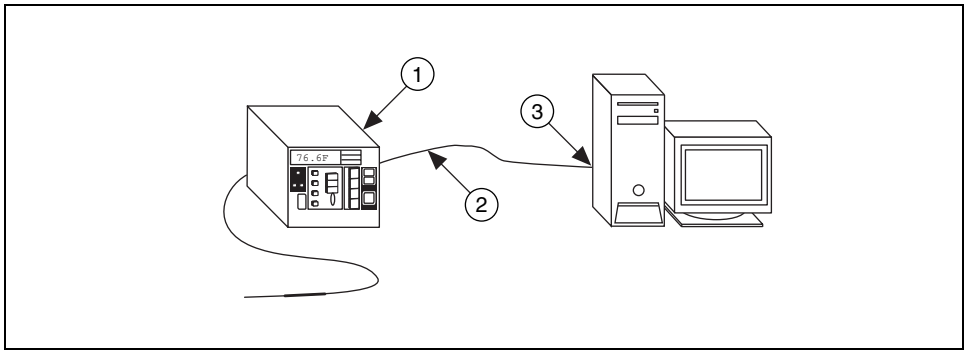
To achieve the high data transfer rate that the GPIB was designed for, you must limit the number of devices on the bus and the physical distance between devices.

You can obtain faster data rates with HS488 devices and controllers. HS488 is an extension to GPIB that most NI controllers support.

Note Refer to the National Instruments GPIB support Web site at `ni.com/support/gpibsupp.htm` for more information about GPIB.

C. Serial Port Communication

Serial communication transmits data between a computer and a peripheral device, such as a programmable instrument or another computer. Serial communication uses a transmitter to send data one bit at a time over a single communication line to a receiver. Use this method when data transfer rates are low or you must transfer data over long distances. Most computers have one or more serial ports, so you do not need any extra hardware other than a cable to connect the instrument to the computer or to connect two computers to each other.



1 RS-232 Instrument 2 RS-232 Cable 3 Serial Port

Figure 1. Serial Instrument Example

You must specify four parameters for serial communication: the baud rate of the transmission, the number of data bits that encode a character, the sense of the optional parity bit, and the number of stop bits. A character frame packages each transmitted character as a single start bit followed by the data bits.

Baud rate is a measure of how fast data moves between instruments that use serial communication.

Data bits are transmitted upside down and backwards, which means that inverted logic is used and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left and read 1 for negative voltage and 0 for positive voltage.

An optional parity bit follows the data bits in the character frame. The parity bit, if present, also follows inverted logic. This bit is included as a means of error checking. You specify ahead of time for the parity of the transmission to be even or odd. If you choose for the parity to be odd, the parity bit is set in such a way so the number of 1s add up to make an odd number among the data bits and the parity bit.

The last part of a character frame consists of 1, 1.5, or 2 stop bits that are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, begins with a start bit of positive (SPACE) voltage.

The following figure shows a typical character frame encoding the letter m.

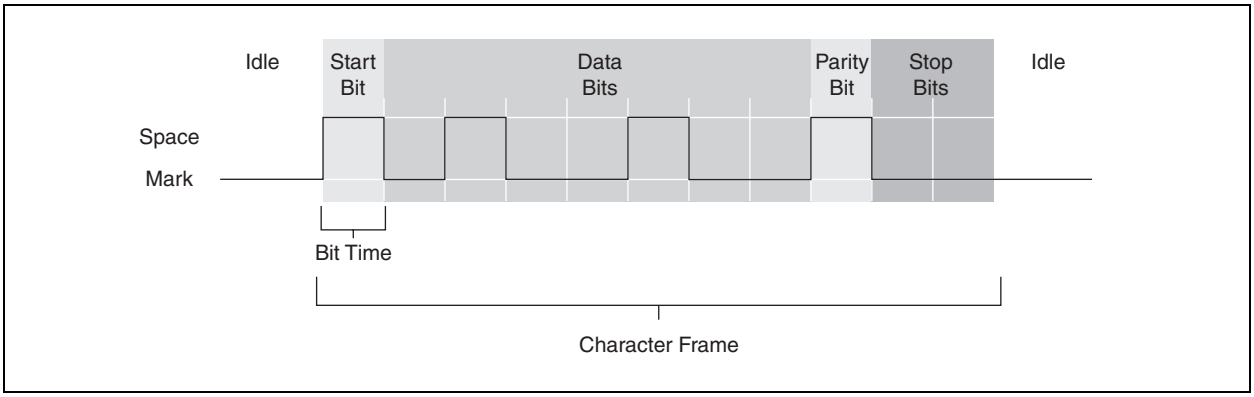


Figure 2. Character Frame for the Letter m

RS-232 uses only two voltage states, called MARK and SPACE. In such a two-state coding scheme, the baud rate is identical to the maximum number of bits of information, including control bits, that are transmitted per second.

MARK is a negative voltage, and SPACE is positive. The previous illustration shows how the idealized signal looks on an oscilloscope. The following is the truth table for RS-232:

Signal > +3 V = 0

Signal < –3 V = 1

The output signal level usually swings between +12 V and –12 V. The dead area between +3 V and –3 V is designed to absorb line noise.

A start bit signals the beginning of each character frame. It is a transition from negative (MARK) to positive (SPACE) voltage. Its duration in seconds is the reciprocal of the baud rate. If the instrument is transmitting at 9,600 baud, the duration of the start bit and each subsequent bit is about 0.104 ms. The entire character frame of eleven bits would be transmitted in about 1.146 ms.

Interpreting the data bits for the transmission yields 1101101 (binary) or 6D (hex). An ASCII conversion table shows that this is the letter m.

This transmission uses odd parity. There are five ones among the data bits, already an odd number, so the parity bit is set to 0.

Data Transfer Rate

You can calculate the maximum transmission rate in characters per second for a given communication setting by dividing the baud rate by the bits per character frame.

In the previous example, there are a total of eleven bits per character frame. If the transmission rate is set at 9,600 baud, you get 9,600/11 = 872 characters per second. Notice that this is the maximum character transmission rate. The hardware on one end or the other of the serial link might not be able to reach these rates, for various reasons.

Serial Port Standards

The following examples are the most common recommended standards of serial port communication:

1. RS-232 (ANSI/EIA-232 Standard) is used for many purposes, such as connecting a mouse, printer, or modem. It also is used with industrial instrumentation. Because of improvements in line drivers and cables, applications often increase the performance of RS-232 beyond the distance and speed in the standards list. RS-232 is limited to point-to-point connections between PC serial ports and devices.
2. RS-422 (AIA RS-422A Standard) uses a differential electrical signal as opposed to the unbalanced (single-ended) signals referenced to ground with RS-232. Differential transmission, which uses two lines each to transmit and receive signals, results in greater noise immunity and longer transmission distances as compared to RS-232.
3. RS-485 (EIA-485 Standard) is a variation of RS-422 that allows you to connect up to 32 devices to a single port and define the necessary electrical characteristics to ensure adequate signal voltages under maximum load. With this enhanced multidrop capability, you can create networks of devices connected to a single RS-485 serial port. The noise immunity and multidrop capability make RS-485 an attractive choice in industrial applications that require many distributed devices networked to a PC or other controller for data collection and other operations.

D. Using Other Interfaces

There are devices made to communicate with serial or GPIB instruments through the Ethernet, USB, or IEEE 1394 (FireWire®) ports, which bypasses the need for a serial port or GPIB board on your computer. When using these devices, program them just as you would if they were using the serial port or a GPIB board.

USB and ethernet interfaces transform USB ports or ethernet ports into asynchronous serial ports for communication with serial instruments. You can install and use these interfaces as standard serial ports from your existing applications.

USB, ethernet, and IEEE 1394 controllers transform any computer with these ports into a full-function, Plug and Play, IEEE-488.2 Controller that can control up to 14 programmable GPIB instruments.

E. Software Architecture

The software architecture for instrument control using LabVIEW is similar to the architecture for DAQ. Instrument interfaces such as GPIB include a set of drivers. Use MAX to configure the interface. VISA, Virtual Instrument Software Architecture, is a common API to communicate with the interface drivers and is the preferred method used when programming for instrument control in LabVIEW, because VISA abstracts the type of interface used. Many LabVIEW VIs used for instrument control use the VISA API. For example, the Instrument I/O Assistant is a LabVIEW Express VI that can use VISA to communicate with message-based instruments and convert the response from raw data to an ASCII representation. Use the Instrument I/O Assistant when an instrument driver is not available. In LabVIEW, an instrument driver is a set of VIs specially written to communicate with an instrument.

Note GPIB drivers are available on the LabVIEW Installer CD-ROM and most GPIB drivers are available for download at `ni.com/support/gpib/versions.htm`. Always install the newest version of these drivers unless otherwise instructed in the release notes.

MAX (Windows; GPIB)

(Windows) Use MAX to configure and test the GPIB interface. MAX interacts with the various diagnostic and configuration tools installed with the driver and also with the Windows Registry and Device Manager. The driver-level software is in the form of a DLL and contains all the functions that directly communicate with the GPIB interface. The Instrument I/O VIs and functions directly call the driver software.

Note **(Mac OS and UNIX)** Refer to documentation supplied with your GPIB interface device for information about configuring and testing the interface.

Open MAX by double-clicking the icon on the desktop or by selecting **Tools»Measurement & Automation Explorer** in LabVIEW. The following example shows a GPIB interface in MAX after clicking the **Scan For Instruments** button on the toolbar.

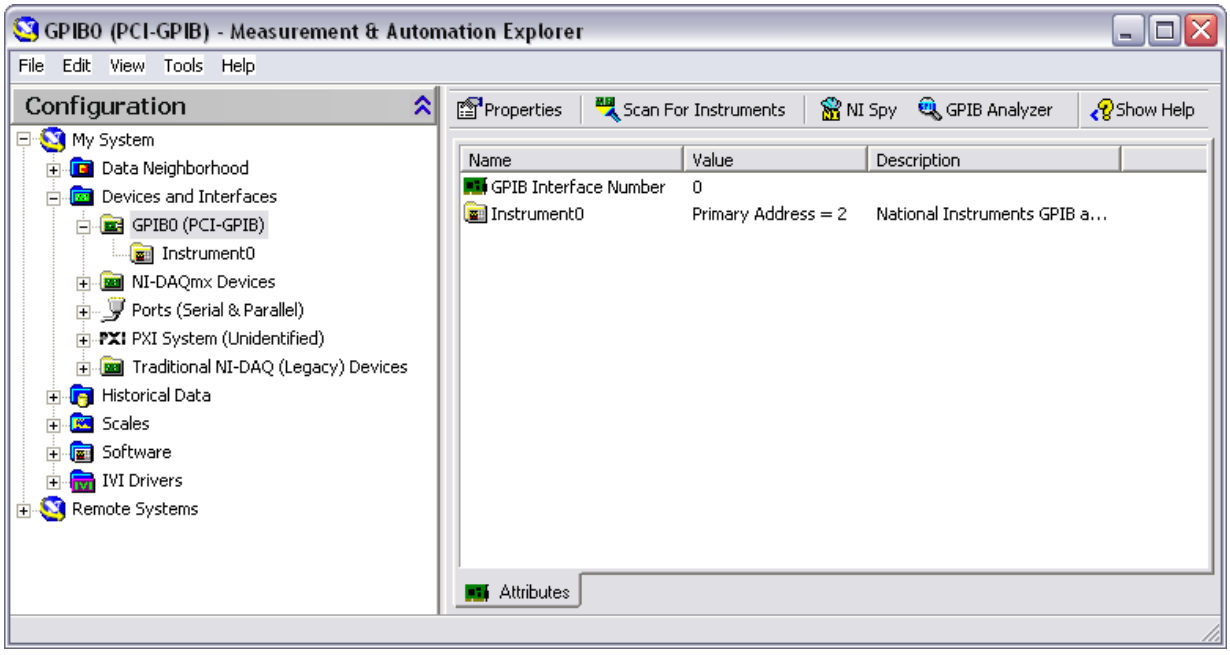


Figure 3. GPIB Interface in Measurement and Automation Explorer

Configure the objects listed in MAX by right-clicking each item and selecting an option from the shortcut menu. You learn to use MAX to configure and communicate with a GPIB instrument in the next exercise.

Exercise 1 Concept: GPIB Configuration with MAX

Goal

Learn to configure the NI Instrument Simulator and use MAX to examine the GPIB interface settings, detect instruments, and communicate with an instrument.

Description

1. Configure the NI Instrument Simulator.
   - ❑ Power off the NI Instrument Simulator.
   - ❑ Set the left bank of switches on the side of the box to match Figure 4.
   - ❑ Power on the NI Instrument Simulator.
   - ❑ Verify that both the Power and Ready LEDs are lit.

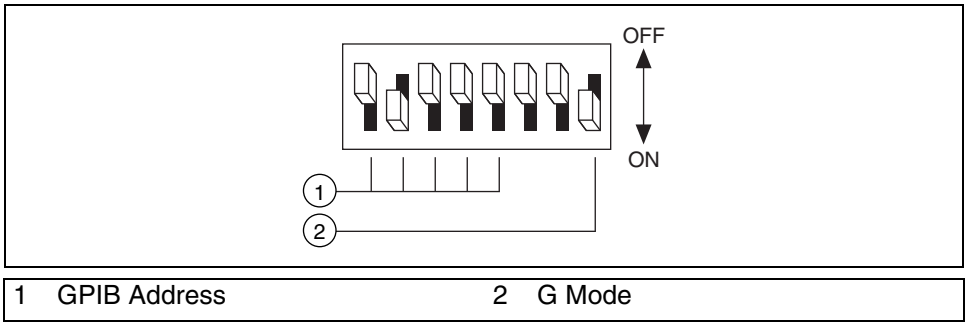


Figure 4. GPIB Configuration Settings for the NI Instrument Simulator

2. Launch MAX by either double-clicking the icon on the desktop or by selecting **Tools»Measurement & Automation Explorer** in LabVIEW.
3. View the settings for the GPIB interface.
   - ❑ Expand the **Devices and Interfaces** section to display the installed interfaces. If a GPIB interface is listed, the NI-488.2 software is correctly loaded on the computer.
   - ❑ Select the GPIB interface and click the **Properties** button on the toolbar to display the **Properties** dialog box.
   - ❑ Examine but do not change the settings for the GPIB interface.
   - ❑ Click the **OK** button to close the dialog box.

4. Communicate with the GPIB instrument.

   ❑ Make sure the GPIB interface is still selected in the **Devices and Interfaces** section.

   ❑ Click the **Scan for Instruments** button on the toolbar.

   ❑ Expand the GPIB interface that is selected in the **Devices and Interfaces** section. One instrument named `Instrument0` appears.

   ❑ Click **Instrument0** to display information about it in the right pane of MAX. Notice that the NI Instrument Simulator has a GPIB primary address (PAD) of 2.

   ❑ Click the **Communicate with Instrument** button on the toolbar. An interactive window appears. You can use it to query, write to, and read from that instrument.

   ❑ Enter `*IDN?` in **Send String** and click the **Query** button. The instrument returns its make and model number in **String Received** as shown in Figure 5. You can use this window to debug instrument problems or to verify that specific commands work as described in the instrument documentation.

Figure 5. Communication with the GPIB Instrument

   ❑ Enter `MEAS:DC?` in **Send String** and click the **Query** button. The NI Instrument Simulator returns a simulated voltage measurement.

   ❑ Click the **Query** button again to return a different value.

   ❑ Click the **Exit** button when done.

5. Set a VISA alias of `devsim` for the NI Instrument Simulator so you can use the alias instead of having to remember the primary address.

   - ❑ While **Instrument0** is selected in MAX, select the **VISA Properties** tab.
   - ❑ Enter `devsim` in the **VISA Alias on My System** field. You will use this alias throughout this lesson.

6. Select **File»Exit** to exit MAX.
7. Click **Yes** when prompted to save the instrument.

End of Exercise 1

F. Instrument I/O Assistant

The Instrument I/O Assistant is a LabVIEW Express VI which you can use to communicate with message-based instruments and convert the response from raw data to an ASCII representation. You can communicate with an instrument that uses a serial, Ethernet, or GPIB interface. Use the Instrument I/O Assistant when an instrument driver is not available.

The Instrument I/O Assistant organizes instrument communication into ordered steps. To use Instrument I/O Assistant, you place steps into a sequence. As you add steps to the sequence, they appear in the **Step Sequence** window. Use the view associated with a step to configure instrument I/O.

To launch the Instrument I/O Assistant, place the Instrument I/O Assistant Express VI on the block diagram in LabVIEW. The Instrument I/O Assistant Express VI is available in the Instrument I/O category of the Functions palette. The **Instrument I/O Assistant** configuration dialog box appears. If it does not appear, double-click the Instrument I/O Assistant icon. Complete the following steps to configure the Instrument I/O Assistant.

1. Select an instrument. Instruments that have been configured in MAX appear in the **Select an instrument** pull-down menu.
2. Choose a **Code generation type**. VISA code generation allows for more flexibility and modularity than GPIB code generation.
3. Select from the following communication steps using the **Add Step** button:
   - **Query and Parse**—Sends a query to the instrument, such as `*IDN?` and parses the returned string. This step combines the Write command and Read and Parse command.
   - **Write**—Sends a command to the instrument.
   - **Read and Parse**—Reads and parses data from the instrument
4. After adding the desired number of steps, click the **Run** button to test the sequence of communication that you have configured for the Express VI.
5. Click the **OK** button to exit the **Instrument I/O Assistant** configuration dialog box.

LabVIEW adds input and output terminals to the Instrument I/O Assistant Express VI on the block diagram that correspond to the data you receive from the instrument.

To view the code generated by the Instrument I/O Assistant, right-click the Instrument I/O Assistant icon and select **Open Front Panel** from the shortcut menu. This converts the Express VI to a subVI. Switch to the block diagram to see the code generated.

Note After you convert an Express VI to a subVI, you cannot reconvert the Express VI.

Exercise 2 Concept: Instrument I/O Assistant

Goal

Configure a serial or GPIB instrument and communicate with the instrument using the Instrument I/O Assistant.

For serial, follow the instructions in part A of this exercise.

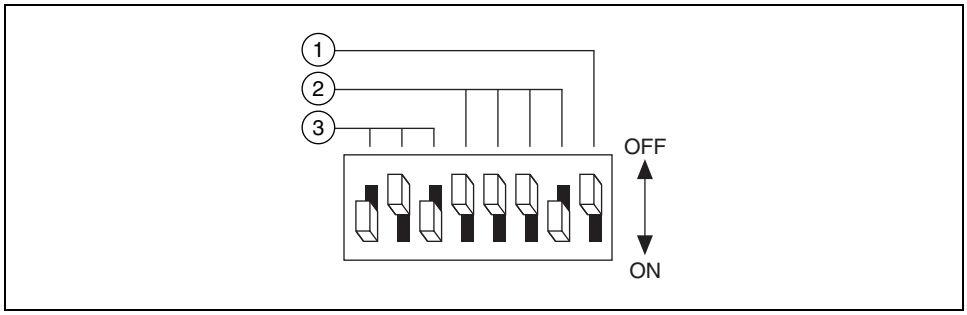
For GPIB, follow the instructions in part B of this exercise.

Part A: Serial Description

1. Configure the NI Instrument Simulator to communicate through the serial port.

   ❑ Power off the NI Instrument Simulator.

   ❑ Set the left bank of switches on the side of the box to match Figure 6.



1 S Mode	2 Data Format	3 Baud Rate

Figure 6. Serial Configuration Settings for the NI Instrument Simulator

   ❑ Make sure the NI Instrument Simulator is connected to a serial port on the computer with a serial cable.

   ❑ Make a note of the port number.

   ❑ Power on the NI Instrument Simulator.

   ❑ Verify that the Power, Ready, and Listen LEDs are lit to indicate that the device is in serial communication mode.

You build a block diagram similar to the one in Figure 7 in the following steps.

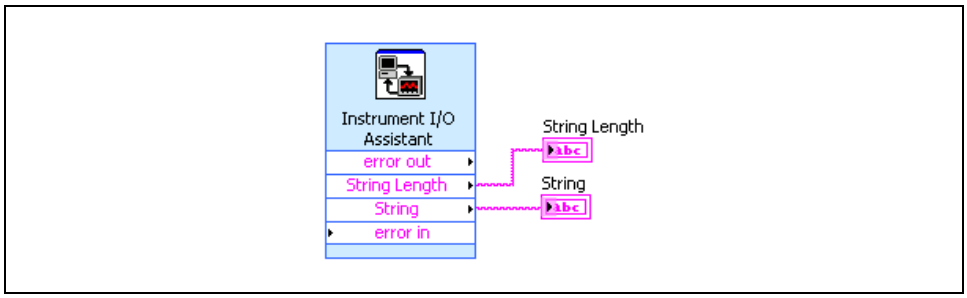


Figure 7. IIOASerial VI Block Diagram

2. Open a blank VI.
3. Save the VI as `Serial IIOA Read.vi` in the `C:\Exercises\LabVIEW Basics I\Instrument IO Assistant` directory.
4. Open the block diagram.
5. Configure the Instrument I/O Express VI to communicate with the NI Instrument Simulator.

   ❑ Place the Instrument I/O Express VI on the block diagram. The **Instrument I/O Assistant** dialog box appears.

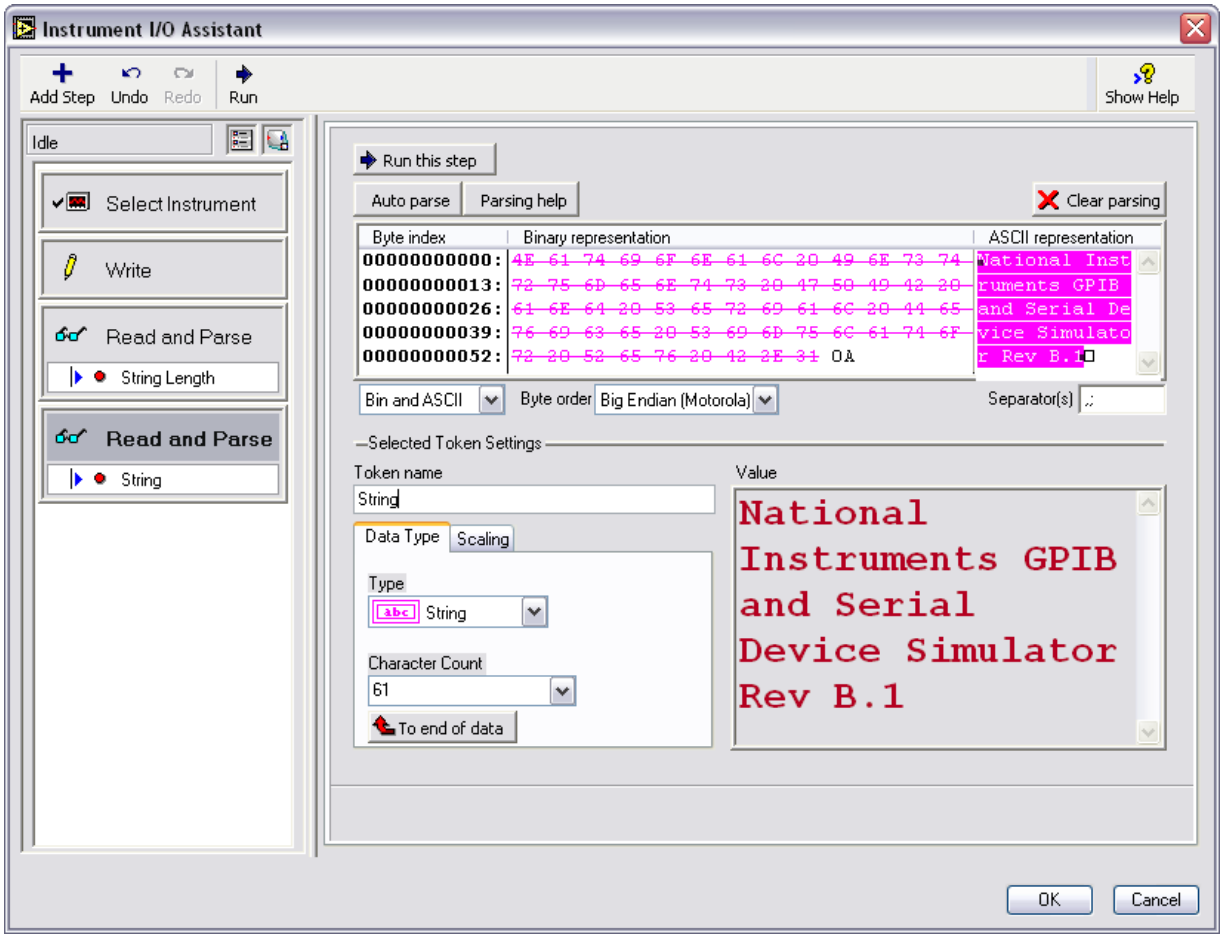


Figure 8. Serial Configuration of the Instrument I/O Assistant

- ❑ From the **Select an instrument** pull-down menu, choose **COM1** (or **COM2** depending on the connection port of the NI Instrument Simulator).
- ❑ From the **Termination Character** pull-down menu, choose **\n**.
- ❑ Click the **Add Step** button.
- ❑ Click **Write**.
- ❑ In the command field, enter `*IDN?`.
- ❑ Click the **Add Step** button.
- ❑ Click **Read and Parse**.

Note The Instrument Simulator returns the byte size of the response, the termination character, the response, then another termination character. Therefore, after `*IDN?` is sent to the instrument, the response must be read twice; once to retrieve the size of the response, and once to retrieve the response.

- ❑ Click the **Add Step** button.
- ❑ Click **Read and Parse** again.
- ❑ Click the **Run** button (not the **Run this step** button). The **Run** button runs the entire sequence.
- ❑ Return to the first **Read and Parse** step.
- ❑ Click the **Auto parse** button. The value returned is the size in bytes of the query response.
- ❑ Rename `Token` to `String Length` in the **Token name** text box.
- ❑ Select the second **Read and Parse** step.
- ❑ Click the **Auto parse** button. The value returned is the identification string of the NI Instrument Simulator.
- ❑ Rename `Token` to `String` in the **Token name** text box. The configuration window should be similar to Figure 8.
- ❑ Select **OK** to return to the block diagram.

6. Create an indicator for the response from the instrument.
   - ❑ Right-click the **String** terminal.
   - ❑ Select **Create»Indicator** from the shortcut menu.
7. Create an indicator for the response length from the instrument.
   - ❑ Right-click the **String Length** terminal.
   - ❑ Select **Create»Indicator** from the shortcut menu.

Tip To allow LabVIEW to handle errors automatically, do not connect a Simple Error Handler VI to **error out**.

8. Display the front panel. It should be similar to Figure 9.

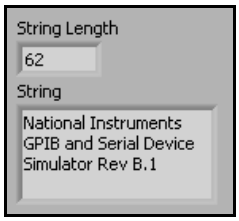


Figure 9. IIOASerial VI Front Panel

9. Save the VI.

10. Run the VI.

11. Examine the code generated by the I/O Assistant.

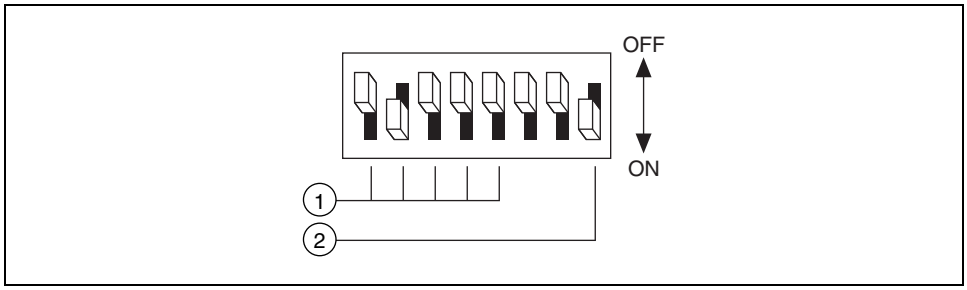
    - ❑ Right-click the I/O Assistant and select **Open Front Panel**.
    - ❑ Click the **Convert** button when asked if you want to convert to a subVI.
    - ❑ View the code generated by the I/O Assistant. Where is the command `*IDN?` written to the Instrument Simulator?
    - ❑ Select **File»Exit** to exit the subVI. Do not save changes.

12. Close the VI when finished.

Part B: GPIB Description

1. Configure the NI Instrument Simulator to communicate through the GPIB interface.

    - ❑ Power off the NI Instrument Simulator.
    - ❑ Set the left bank of switches on the side of the box to match Figure 10.



1 GPIB Address 2 G Mode

Figure 10. GPIB Configuration Settings for the NI Instrument Simulator

    - ❑ Make sure the NI Instrument Simulator is connected to the GPIB board.
    - ❑ Power on the NI Instrument Simulator.
    - ❑ Verify that both the Power and Ready LEDs are lit.

2. Open a blank VI.

3. Save the VI as `GPIB IIOA Read.vi` in the `C:\Exercises\LabVIEW Basics I\Instrument IO Assistant` directory.

You build a block diagram similar to the one in Figure 11 in the following steps.

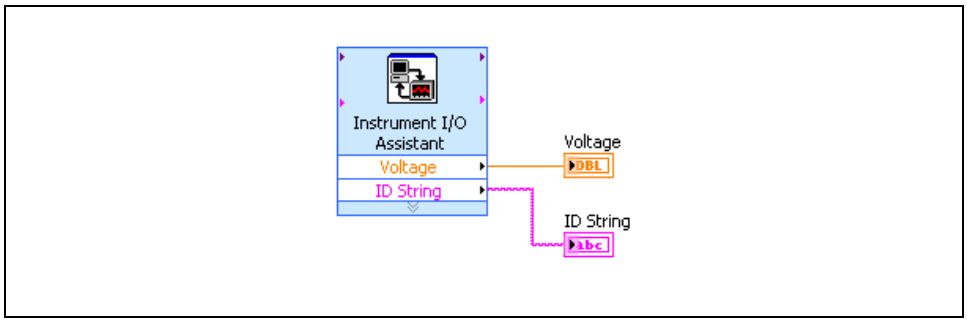


Figure 11. IIOAGPIB VI Block Diagram

4. Open the block diagram.

5. Configure the Instrument I/O Express VI to communicate with the NI Instrument Simulator.

❑ Place the Instrument I/O Express VI on the block diagram. The **Instrument I/O Assistant** dialog box appears.

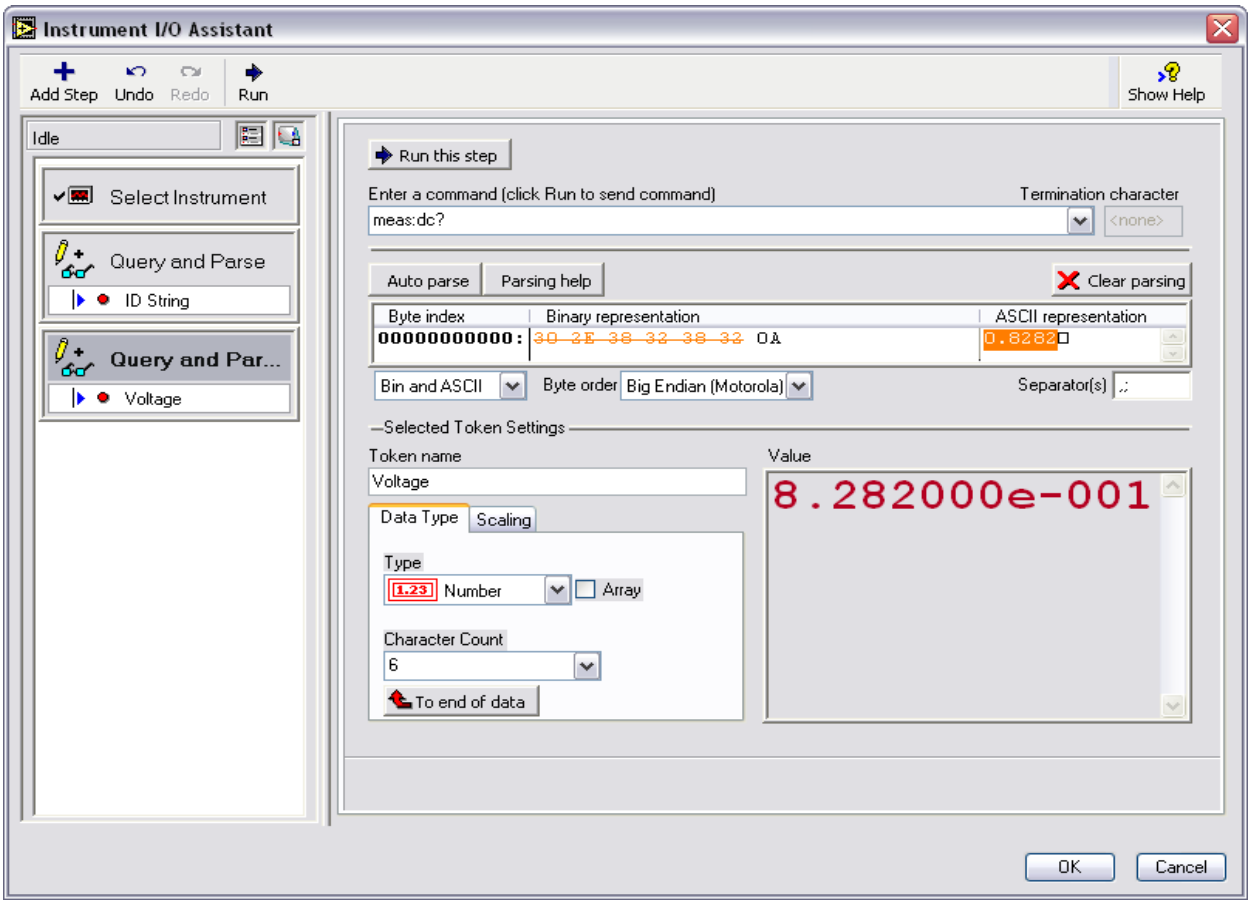


Figure 12. GPIB Configuration of the Instrument I/O Assistant

- ❑ Select **devsim** from the **Select an instrument** pull-down menu.
- ❑ Select **VISA Code Generation** from the **Code generation type** pull-down menu.
- ❑ Click the **Add Step** button.
- ❑ Click **Query and Parse** to write and read from the Instrument Simulator.
- ❑ Enter `*IDN?` as the command.
- ❑ Click the **Run this step** button. If no error warning appears in the lower half of the dialog box, this step has successfully completed.
- ❑ To parse the data received, click the **Auto parse** button.
- ❑ Rename `Token` by typing `ID String` in the **Token name** text box.
- ❑ Click the **Add Step** button.
- ❑ Click **Query and Parse**.
- ❑ Enter `MEAS:DC?` as the command.
- ❑ Click the **Run this step** button.
- ❑ To parse the data received, click the **Auto parse** button. The data returned is a random numeric value.
- ❑ Rename `Token` by typing `Voltage` in the **Token name** text box. The configuration window should be similar to Figure 12.
- ❑ Click the **OK** button to exit the I/O Assistant and return to the block diagram.

6. Create an indicator for the ID String.
   - ❑ Right-click the **ID String** terminal and select **Create»Indicator** from the shortcut menu.
7. Create an indicator for the voltage.
   - ❑ Right-click the **Voltage** terminal and select **Create»Indicator** from the shortcut menu.

Tip To allow LabVIEW to handle errors automatically, do not connect a Simple Error Handler VI to **error out**.

8. Display the front panel. The front panel should be similar to the front panel in Figure 13.

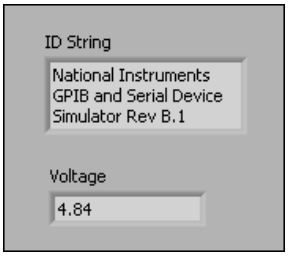


Figure 13. IIOAGPIB VI Front Panel

9. Save the VI.

10. Run the VI. Resize the string indicator if necessary.

11. Examine the code generated by the I/O Assistant.

    - ❑ Right-click the I/O Assistant and select **Open Front Panel**.
    - ❑ Click the **Convert** button when prompted to convert to a subVI.
    - ❑ View the code generated by the I/O Assistant. Where is the command `*IDN?` written to the Instrument Simulator? Where is the voltage being read?
    - ❑ Select **File»Exit** to exit the subVI. Do not save changes.

12. Close the VI when finished.

End of Exercise 2

G. VISA

Virtual Instrument Software Architecture (VISA) is the lower layer of functions in the LabVIEW instrument driver VIs that communicates with the driver software. VISA by itself does not provide instrumentation programming capability. VISA is a high-level API that calls low-level drivers. VISA can control VXI, GPIB, serial, or computer-based instruments and makes the appropriate driver calls depending on the type of instrument used. When debugging VISA problems, remember that an apparent VISA problem could be an installation problem with one of the drivers that VISA calls.

In LabVIEW, VISA is a single library of functions you use to communicate with GPIB, serial, VXI, and computer-based instruments. You do not need to use separate I/O palettes to program an instrument. For example, some instruments give you a choice for the type of interface. If the LabVIEW instrument driver were written with functions on the **Functions»All Functions»Instrument I/O»GPIB** palette, those instrument driver VIs would not work for the instrument with the serial port interface.
VISA solves this problem by providing a single set of functions that work for any type of interface. Therefore, many LabVIEW instrument drivers use VISA as the I/O language.

VISA Programming Terminology

The following terminology is similar to that used for instrument driver VIs:

- **Resource**—Any instrument in the system, including serial and parallel ports.
- **Session**—You must open a VISA session to a resource to communicate with it, similar to a communication channel. When you open a session to a resource, LabVIEW returns a VISA session number, which is a unique refnum to that instrument. You must use the session number in all subsequent VISA functions.
- **Instrument Descriptor**—Exact name of a resource. The descriptor specifies the interface type (GPIB, VXI, ASRL), the address of the device (logical address or primary address), and the VISA session type (INSTR or Event).

The instrument descriptor is similar to a telephone number, the resource is similar to the person with whom you want to speak, and the session is similar to the telephone line. Each call uses its own line, and crossing these lines results in an error. Table 1 shows the proper syntax for the instrument descriptor.

Table 1. Syntax for Various Instrument Interfaces

Interface	Syntax
Asynchronous serial	`ASRL[`board`][::INSTR]`
GPIB	`GPIB[`board`]::`primary address`[::`secondary address`][::INSTR]`
VXI instrument through embedded or MXIbus controller	`VXI[`board`]::`VXI logical address`[::INSTR]`
GPIB-VXI controller	`GPIB-VXI[`board`][::`GPIB-VXI primary address`::`VXI logical address`[::INSTR]`

You can use an alias you assign in MAX instead of the instrument descriptor. **(Mac OS)** Edit the `visaconf.ini` file to assign a VISA alias. **(UNIX)** Use the `visaconf` utility.

If you choose not to use the Instrument I/O Assistant to automatically generate code for you, you can still write a VI to communicate with the instrument. The most commonly used VISA communication functions are the VISA Write and VISA Read functions. Most instruments require you to send information in the form of a command or query before you can read information back from the instrument. Therefore, the VISA Write function is usually followed by a VISA Read function. The VISA Write and VISA Read functions work with any type of instrument communication and are the same whether you are doing GPIB or serial communication. However, because serial communication requires you to configure extra parameters, you must start the serial port communication with the VISA Configure Serial Port VI.

VISA and Serial

The VISA Configure Serial Port VI initializes the port identified by **VISA resource name** to the specified settings. **Timeout** sets the timeout value for the serial communication. **Baud rate**, **data bits**, **parity**, and **flow control** specify those specific serial port parameters. The **error in** and **error out** clusters maintain the error conditions for this VI.

Figure 14 shows how to send the identification query command `*IDN?` to the instrument connected to the COM2 serial port. The VISA Configure Serial Port VI opens communication with COM2 and sets it to 9,600 baud, eight data bits, odd parity, one stop bit, and XON/XOFF software handshaking. Then, the VISA Write function sends the command. The VISA Read function reads back up to 200 bytes into the read buffer, and the Simple Error Handler VI checks the error condition.

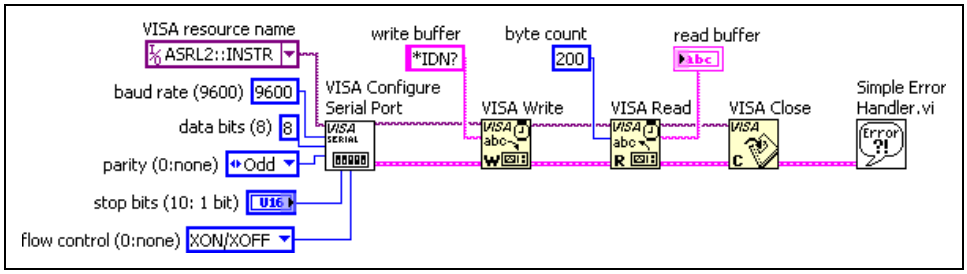


Figure 14. Configuring serial for VISA example

Note The VIs and functions located on the **Functions»All Functions»Instrument I/O» Serial** palette are also used for parallel port communication. You specify the VISA resource name as being one of the LPT ports. For example, you can use MAX to determine that LPT1 has a VISA resource name of `ASRL10::INSTR`.

Exercise 3 VISA Write & Read VI

Goal

Communicate with a serial or GPIB interface to an instrument using VISA functions.

Description

This VI uses VISA to communicate with either a serial or a GPIB interface to an instrument. The VI can send one buffer of data to the instrument, and read one buffer back. If using GPIB, the user specifies how many bytes to read from the bus. If using serial, the VI determines how many bytes are available, and reads them all.

1. Open the `VISA Write & Read.vi` in the `C:\Exercises\LabVIEW Basics I\VISA Write & Read` directory.

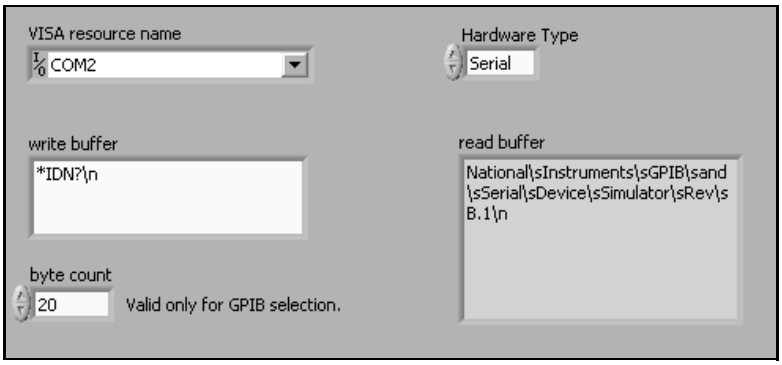


Figure 15. VISA Write & Read VI Front Panel

2. Open the block diagram of the VI and examine the code. The GPIB portion is shown in Figure 16.

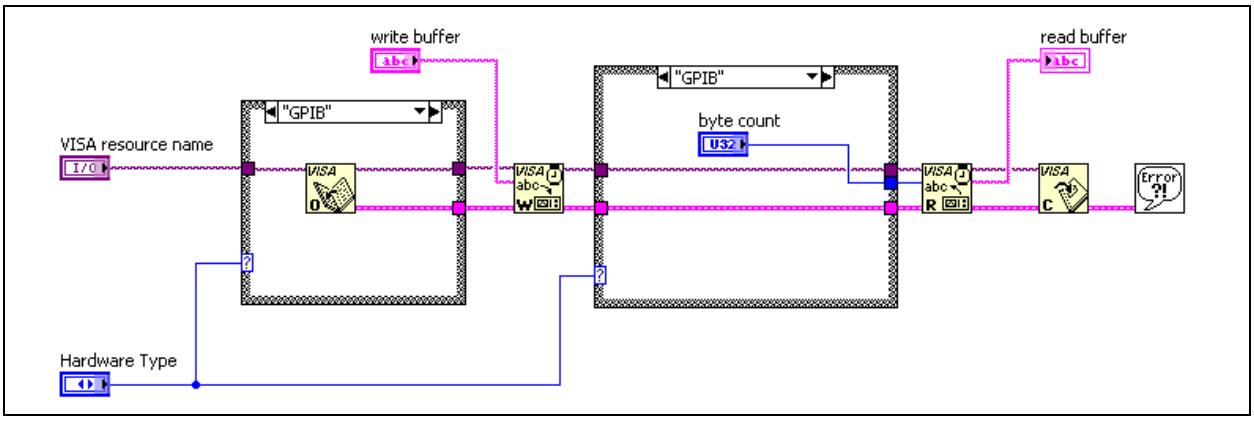


Figure 16. GPIB portion of the VISA Write & Read VI Block Diagram

Follow the instructions in the *Test A: Serial* section to communicate through the serial port. Follow the instructions in the *Test B: GPIB* section to communicate through the GPIB port.

Test A: Serial

1. Configure the NI Instrument Simulator to communicate through the serial port. It may still be set up from the last exercise.

   ❑ Power off the NI Instrument Simulator.

   ❑ Set the left bank of switches on the side of the box to match Figure 17.

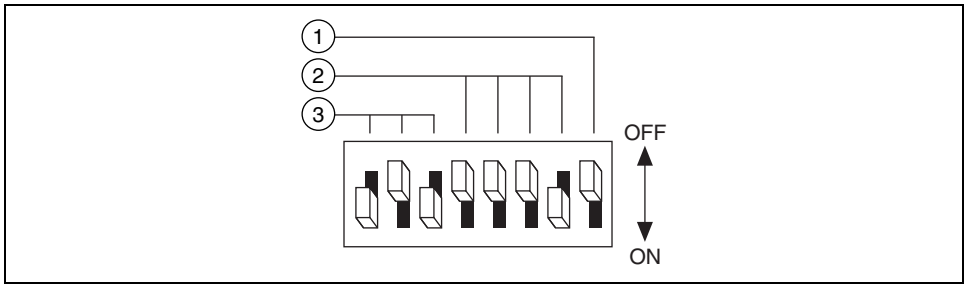


Figure 17. Serial Configuration Settings for the NI Instrument Simulator

   ❑ Make sure the NI Instrument Simulator is connected to a serial port.

   ❑ Power on the NI Instrument Simulator.

   ❑ Verify that the Power, Ready and Listen LEDs are lit.

2. Enter values into the controls in preparation for communicating with the instrument. You do not need to enter a value in the byte count, as this control is only used for GPIB communication.

   ❑ Select the serial port in the **VISA resource name** control.

   ❑ Select Serial from the Hardware Type enumerated control.

   ❑ Enter `*IDN?` in the **write buffer**.

3. Run the VI.

4. The top of the instrument simulator lists other commands that are recognized by this instrument. Try other commands in this VI.

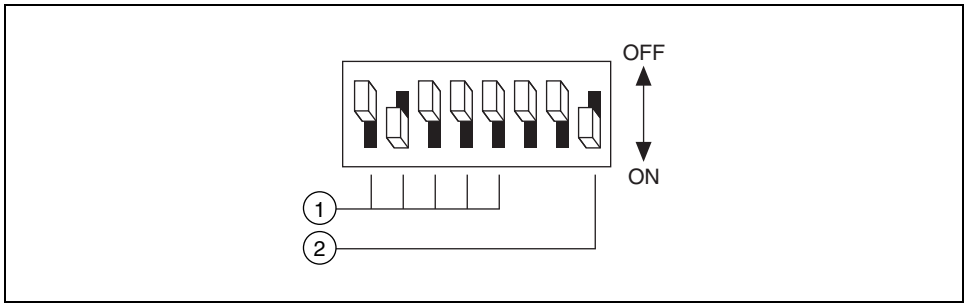
5. Close the VI when finished.

Test B: GPIB

1. Configure the NI Instrument Simulator to communicate through the GPIB interface.

   ❑ Power off the NI Instrument Simulator.

   ❑ Set the left bank of switches on the side of the box to match Figure 18.



1	GPIB Address Set the GPIB Address	2	G Mode GPIB Mode

Figure 18. GPIB Configuration Settings for the NI Instrument Simulator

   ❑ Make sure the NI Instrument Simulator is connected to the GPIB board.

   ❑ Power on the NI Instrument Simulator.

   ❑ Verify that both the Power and Ready LEDs are lit.

2. Enter values into the controls in preparation for communicating with the instrument.

   ❑ Select devsim in the **VISA resource name** control.

   ❑ Select GPIB from the Hardware Type enumerated control.

   ❑ Enter `*IDN?` in the **write buffer**.

3. Run the VI.

4. The top of the instrument simulator lists other commands that are recognized by this instrument. Try other commands in this VI.

5. Close the VI when finished.

End of Exercise 3

H. Instrument Drivers

Imagine the following scenario. You wrote a LabVIEW VI that communicates with a specific oscilloscope in your lab. Unfortunately, the oscilloscope no longer works, and you must replace it. However, this particular oscilloscope is no longer made. You found a different brand of oscilloscope that you want to purchase, but your VI no longer works with the new oscilloscope. You must rewrite your VI.

When you use an instrument driver, the driver contains the code specific to the instrument. Therefore, if you change instruments, you must replace only the instrument driver VIs with the instrument driver VIs for the new instrument, which greatly reduces your redevelopment time. Instrument drivers help make test applications easier to maintain because the drivers contain all the I/O for an instrument in one library, separate from other code. When you upgrade hardware, upgrading the application is easier because the instrument driver contains all the code specific to that instrument.

What Is an Instrument Driver?

A LabVIEW Plug and Play instrument driver is a set of VIs that control a programmable instrument. Each VI corresponds to an instrument operation, such as configuring, triggering, and reading measurements from the instrument. Instrument drivers help users get started using instruments from a PC and saves them development time and cost because users do not need to learn the programming protocol for each instrument. With open-source, well documented instrument drivers, end users can customize their operation for better performance. A modular design makes the driver easier to customize.

Where Do I Find Instrument Drivers?

You can locate most LabVIEW Plug and Play instrument driver in the Instrument Driver Finder. You can access the Instrument Driver Finder within LabVIEW by selecting **Tools»Instrumentation»Find Instrument Drivers** or **Help»Find Instrument Drivers**. The Instrument Driver Finder connects you with `ni.com` to find instrument drivers. When you install an instrument driver, an example program using the driver is added to the NI Example Finder.

Example Instrument Driver VI

The block diagram in Figure 19 initializes the Agilent 34401 digital multimeter (DMM), uses a configuration VI to choose the resolution and range, select the function, and enable or disable auto range, uses a data VIs to read a single measurement, closes the instrument, and checks the error status. Every application that uses an instrument driver has a similar sequence of events: Initialize, Configure, Data, and Close.

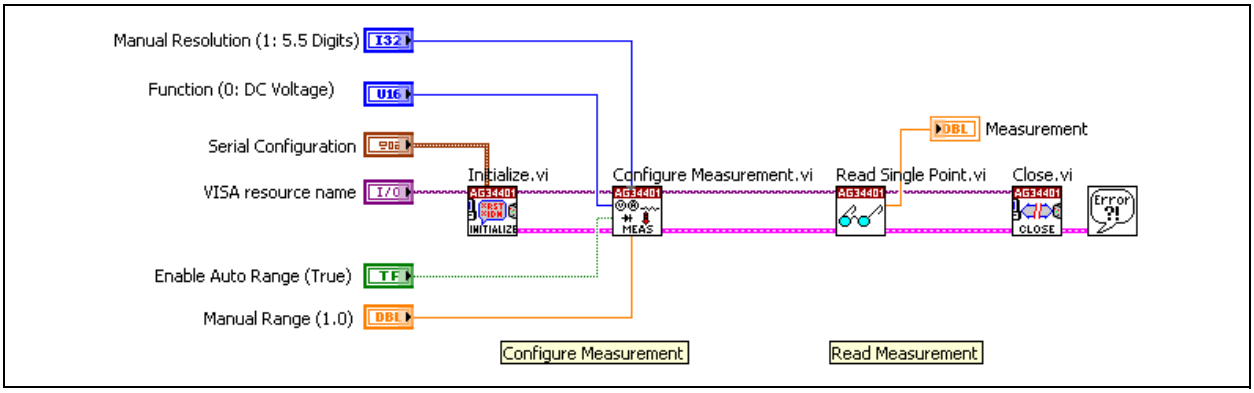


Figure 19. Agilent 34401 DMM Instrument Driver Example

This is an example program that is available in the NI Example Finder when you install the LabVIEW Plug and Play instrument driver for the Agilent 34401 DMM.

How Do Instrument Drivers Work?

Many programmable instruments have a large number of functions and modes. With this complexity, it is necessary to provide a consistent design model that aids both instrument driver developers as well as end users who develop instrument control applications. The LabVIEW Plug and Play instrument driver model contains both external structure and internal structure guidelines. The external structure shows how the instrument driver interfaces with the user and to other software components in the system. The internal structure shows the internal organization of the instrument driver software module.

For the external structure of the instrument driver, the user interacts with the instrument driver using an API or an interactive interface. Usually, the interactive interface is used for testing or for end-users. The API is accessed through LabVIEW. The instrument driver communicates with the instrument using VISA.

Internally, the VIs in an instrument driver are organized into six categories. These categories are summarized in the following table.

Category	Description
Initialize	The initialize VI establishes communication with the instrument and is the first instrument driver VI called.
Configure	This collection of VIs are software routines that configure the instrument to perform specific operations. After calling these VIs, the instrument is ready to take measurements or stimulate a system.

Category	Description
Action/Status	This collection of VIs command the instrument to carry out an action (i.e. arming a trigger) or obtain the current status of the instrument or pending operations.
Data	The data VIs transfer data to or from the instrument.
Utility	THis collection of VIs perform a variety of auxiliary operations, such as reset and self-test.
Close	The close VI terminates the software connection to the instrument. This is the last instrument driver VI called.

Exercise 4 Concept: Instrument Driver 

Goal

Install an instrument driver and explore the example programs that accompany the instrument driver.

Description

In this exercise, you install the instrument driver for the NI Instrument Simulator. After installation, you explore the VIs that the instrument driver provides and the example programs that are added to the NI Example Finder.

Install Instrument Driver

1. Exit LabVIEW.
2. Navigate to the `C:\Exercises\LabVIEW Basics I\Instrument Driver` directory. This folder contains the LabVIEW Plug and Play instrument driver for the Instrument Simulator.
3. Double-click the NI Instrument Simulator Zip folder to extract the contents.
4. Copy the folder that was extracted.
5. Navigate to the `C:\Program Files\National Instruments\LabVIEW 8.0\instr.lib` directory.
6. Paste the copied folder in this directory.

Explore Instrument Driver

7. Start LabVIEW.
8. Open a blank VI.
9. Switch to the block diagram.
10. Navigate to the **Instrument I/O»Instrument Drivers»NI Instrument Simulator** category of the **Functions** palette.
11. Explore the palette using the **Context Help** window to familiarize yourself with the functionality.

Use Example Programs

12. Select **Help»Find Examples** to start the NI Example Finder.

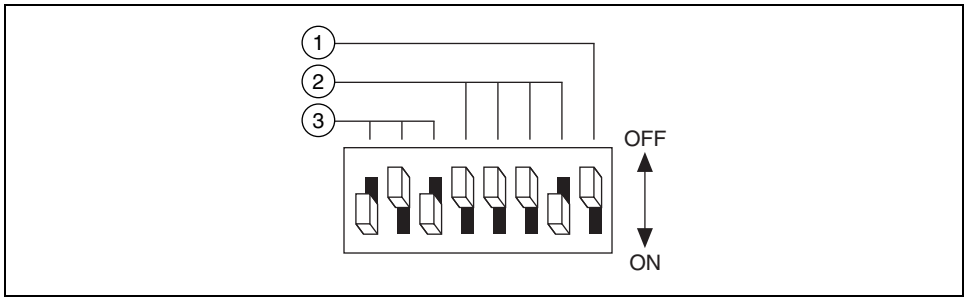
13. Confirm that you are browsing according to task.

14. Navigate to **Hardware Input and Output»Instrument Drivers» LabVIEW Plug and Play** in the task structure.

15. Double-click **NI Instrument Simulator Read DMM Measurement.vi** to open the example program. This VI reads a single measurement from the Instrument Simulator.

16. Prepare the Instrument Simulator. This VI can communicate with the instrument through serial or GPIB.

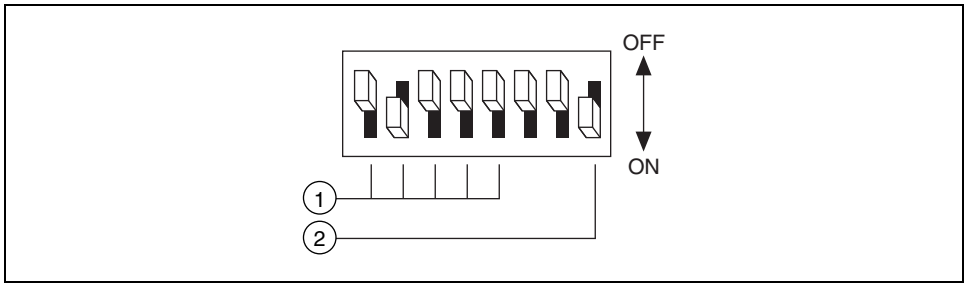
    ❑ To communicate through serial, set the Instrument Simulator switches as shown in Figure 20.



1 S Mode	2 Data Format	3 Baud Rate

Figure 20. Serial Configuration Settings for the NI Instrument Simulator

    ❑ To communicate through GPIB, set the Instrument Simulator switches as shown in Figure 21.



1 GPIB Address Set the GPIB Address	2 G Mode GPIB Mode

Figure 21. GPIB Configuration Settings for the NI Instrument Simulator

17. Select the communication type on the VISA Resource Name control.

    - ❑ If you are using serial, select the resource (COM1 or COM2) that the serial cable is connected to.
    - ❑ If you are using GPIB, select the devsim VISA alias.

18. Run the VI.
19. Explore the block diagram of the VI. Do not save changes.
20. Close the VI.
21. Return to the NI Example Finder.
22. Double-click **NI Instrument Simulator Read Oscilloscope Waveform.vi** to open the next example program. This VI read a single waveform from the Instrument Simulator.
23. Select the same VISA Resource Name you selected in step 17.
24. Run the VI.
25. Choose a different Waveform Function.
26. Run the VI again.
27. Explore the block diagram of the VI.
28. Close the VI and the NI Example Finder when you are finished. Do not save changes.

End of Exercise 4

Self Review: Quiz

1. Which instrument interface does not use the VISA API?
   a. Serial
   b. DAQ
   c. GPIB
   d. Ethernet

2. What API does the Instrument I/O Assistant use?
   a. C
   b. Visual Basic
   c. VISA
   d. NI-DAQmx

3. Which of the following is a way to inform listeners that all data has been transferred?
   a. Asserting the End or Identify (EOI) line.
   b. Placing a end-of-string (EOS) character at the beginning of the data being transferred.
   c. Using the VISA Close function.
   d. Turning off the power to the controller.

Self Review: Quiz Answers

1. Which instrument interface does not use the VISA API?
   a. Serial
   b. DAQ
   c. GPIB
   d. Ethernet

2. What API does the Instrument I/O Assistant use?
   a. C
   b. Visual Basic
   c. VISA
   d. NIDAQmx

3. Which of the following is a way to inform listeners that all data has been transferred?
   a. Asserting the End or Identify (EOI) line.
   b. Placing a end-of-string (EOS) character at the beginning of the data being transferred.
   c. Using the VISA Close function.
   d. Turning off the power to the controller.