Computer-Based Instruments

NI-SWITCH™ Software User Manual

NATIONAL INSTRUMENTS™

June 1998 Edition
Part Number 321900A-01

Internet Support
E-mail: `support@natinst.com`
FTP Site: `ftp.natinst.com`
Web Address: `http://www.natinst.com`

Bulletin Board Support
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support
512 418 1111

Telephone Support (USA)
Tel: 512 795 8248
Fax: 512 794 5678

International Offices
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway   Austin, Texas 78730-5039   USA   Tel: 512 794 0100

Important Information

Warranty

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

Contents

About This Manual

Organization of This Manual

The *NI-SWITCH Software User Manual* is organized as follows:

- Chapter 1, *Introduction*, discusses how to use this manual, lists what you need to get started, and presents a background of the NI-SWITCH software.
- Chapter 2, *Introductory Programming Examples*, contains examples that use NI-SWITCH to explore some of the basic functionality of your switch hardware. The purpose of these examples is to introduce the programming concepts and to familiarize you with the instrument driver.
- Chapter 3, *Getting Started*, contains an overview of the NI-SWITCH Application Programming Interface (API), defines special terms you need to understand, and introduces the various operations defined by the VXI*plug&play* specifications on instrument drivers.
- Chapter 4, *Manual Switch Control*, describes operations you can use to control a switch card manually—rather than automatically through scanning operations—and discusses the effect of switch topology on manual operations. This chapter also summarizes the reset functions and the various possible levels of reset.
- Chapter 5, *Scanning*, discusses the basic building blocks of scanning, such as scan lists and trigger configuration. In addition, this chapter includes examples of how to do scanning with either the internal software scanning architecture or with hardware scanning.
- Appendix A, *Microsoft Visual Basic Examples*, shows the Visual Basic syntax of the ANSI C examples given earlier in this manual. The examples use the same numbering sequence for easy reference.
- Appendix B, *Multiple Card Scanning*, covers the unique features that affect scanning when you have multiple switch cards wired together to act as one large switch.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <Shift>.
[]	Square brackets enclose optional items—for example, [`response`].
♦	The ♦ symbol indicates that the text following it applies only to a specific product, a specific operating system, or a specific software version.
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
bold	Bold text denotes the names of menus, menu items, and parameters.
bold italic	Bold italic text denotes an important note.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value, as in Windows 3.*x*.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and for statements and comments taken from programs.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace bold italic`	Text in this font denotes the generic names for VI or function calls that are used in LabVIEW, C, or Vision Basic. Because the actual names vary in program code for these operating systems, these generic names are used for simplicity.
`monospace italic`	Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *LabWindows/CVI Instrument Driver Developers Guide*, version 5.0 or later, National Instruments Corporation
- VPP-3.1, *Instrument Drivers Architecture and Design Specification*
- VPP-3.2, *Instrument Driver Functional Body Specification*
- VPP-3.3, *Instrument Driver Interactive Developer Interface Specification*
- VPP-3.4, *Instrument Driver Programmatic Developer Interface Specification*

How to Use This Documentation Set

- Setup and Test—This quick reference steps you through installing the software and hardware, configuring, and a self-test to verify installation.
- Use your switch card user manual to learn about the electrical and mechanical aspects and features of your National Instruments switch card.
- This manual, the *NI-SWITCH Software User Manual*, describes how you can use the NI-SWITCH driver to program your National Instruments switch card.
- Online help—These electronic documents (.HLP) give detailed information on the operations and attributes of NI-SWITCH. There are three help files:
  – LabVIEW
  – LabWindows/CVI and Microsoft Visual C
  – Visual Basic
- `Readme.txt`—Check this file to see if there is any updated information on the hardware and software kit.

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

1 Introduction

This chapter discusses how to use this manual, lists what you need to get started, and presents a background of the NI-SWITCH software.

How to Use This Manual

Use this manual sequentially to learn how to set up a system to use National Instruments switching hardware through the NI-SWITCH driver. Make sure you have all the components described in the next section, *What You Need to Get Started*. Also check the `Readme.txt` file installed with the instrument driver, which includes last-minute changes and updates. Refer to the *Setup and Test* document that came with your hardware for instructions about setting up your system.

After you have set up your system, start with Chapter 2, *Introductory Programming Examples*, to guide yourself through some simple examples. Chapter 3, *Getting Started*, contains some general overview material to introduce some of the concepts used in the driver. Chapter 4, *Manual Switch Control*, and Chapter 5, *Scanning*, contain more in-depth information about the different elements that make up the NI-SWITCH software.

The NI-SWITCH driver is based on the Intelligent Virtual Instrument (IVI) technology from National Instruments and therefore has some enhanced instrument driver features, such as state caching and simulation. Refer to the online help for a description of these features.

What You Need to Get Started

- ❑ Appropriate National Instruments switching hardware, such as the NI 25*xx* series. Refer to the `Readme.txt` file for a list of devices you can use with the version of the driver you have.
- ❑ NI-SWITCH instrument driver software

Background

NI-SWITCH is designed to be an easy-to-use software interface to the line of National Instrument switch products. This driver is based on two industry standards for instrument drivers—VXI*plug&play* and Intelligent Virtual Instruments (IVI).

VXI*plug&play*

The VXI*plug&play* Systems Alliance was formed to solve some of the remaining difficulties in integrating a VXI system. One of the most important components they addressed was that of the instrument driver. The VXI*plug&play* alliance created a standard for instrument drivers and required that any VXI device must have such an instrument driver to be VXI*plug&play* compliant. The NI-SWITCH instrument driver follows this standard by providing an instrument driver based on the VXI*plug&play* standards.

Intelligent Virtual Instruments (IVI)

In 1998, National Instruments introduced an instrument driver model that is VXI*plug&play* compliant, but extended the functionality to many new features users had requested, such as state-caching and simulation modes. This architecture uses a support driver, known as the *IVI Engine*, to handle many of the complex features. The NI-SWITCH instrument driver is fully IVI-compliant and therefore supports the enhancements offered by IVI.

2

Introductory Programming Examples

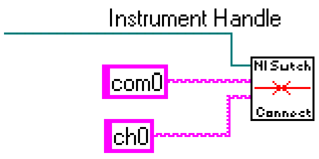
This chapter contains examples that use NI-SWITCH to explore some of the basic functionality of your switch hardware. The purpose of these examples is to introduce the programming concepts and to familiarize you with the instrument driver. Subsequent chapters describe these concepts in more detail.

The examples in the chapters of this manual use LabVIEW and C source code, which you can use in the National Instruments LabVIEW, LabWindows/CVI, and Microsoft Visual C++ programming environments. The NI-SWITCH driver also works with Microsoft Visual Basic for Windows 95/NT programming environments. You can find equivalent examples for this environment in Appendix A, *Microsoft Visual Basic Examples*.

The program shown in Example 2-1 is very straightforward. Later examples in this chapter repeat its basic structure as we introduce additional features.

♦ **C examples**—Any code that is different from Example 2-1 in the subsequent examples appears in **`bold text`**.

Notice also that this manual uses a special font and naming convention for VI or function calls, except in example code itself. For example, the term ***`Connect`***, when shown with this font, refers both to the LabVIEW VI NI-SWITCH Connect:



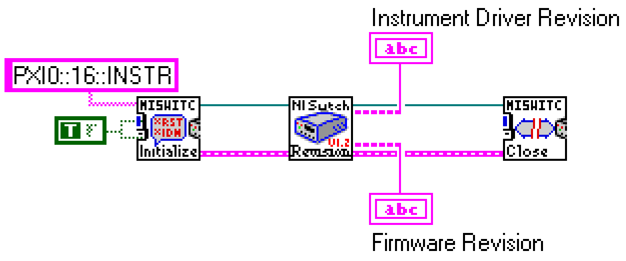
as well as to the C function `niSwitch_Connect()`:

```
niSwitch_Connect(instr, "com0", "ch0");
```

Basic Startup

The NI-SWITCH application-programming interface (API) uses standard initialization and close routines at the beginning and end of the program. Example 2-1 shows, first in LabVIEW and then in C code, how to get a handle to the instrument and retrieve information about the state of the hardware.

Example 2-1 Initialization



```
#include "niswitch.h"

int main (void)
{
    ViSession instr;           /* Communication Channel */
    ViStatus status;           /* For checking errors */
    ViChar firmRev[256];       /* Strings for revision info */
    ViChar driverRev[256];


   /* Begin by opening a communication channel to the instrument */
   status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
   if (status < VI_SUCCESS) {
     /* Error Initializing Interface...exiting */
     return -1;
   }
   /* NOTE: For simplicity, we will not show any other error checking. */


   /* Get the revision of the driver */
   status = niSwitch_revision_query(instr, driverRev, firmRev);


   /* Close communication channel */
   status = niSwitch_close(instr);
   return 0;
}
```

Example 2-1 Discussion

Example 2-1 breaks down into the following steps:

1. Open a communication channel to the device by using ***`Initialize`***. You specify the address of the device you want to talk to through a VISA-style resource string. To use an analogy of telephone communication, this step compares to dialing a phone number. The operation places the call and connects you. The variable returned—**instr**—is a *session*, or communication channel. This represents the connection to the other person on the phone. In this case, it is the connection to the actual hardware.

Note ***This operation is taken directly from the VXI**plug&play* ***specifications.***

   Following the address string are the **ID Query** and **Reset**, which are both Boolean values. If you pass *True* to **ID Query**, the board verifies that the hardware at the specified address is of the correct type (where *type* is defined by the instrument driver—in this case, a National Instruments switch card). If you pass *True* to **Reset**, the driver opens communication with the board and immediately resets the board to a predetermined state—typically the power-on state. Notice that if the board has latching relays, the state is not the last power-on state, but instead the defined power-on state where all relays are in the reset state. These steps do not occur if you pass *False* to the respective inputs.

2. Now that you have successfully established a communication channel, you are ready to perform some action. In this example, you query the driver to check the revision of the driver.

3. At this point, you are done with this example. To finish the communication, you need to close the session. For this purpose, you use another standard VXI*plug&play* operation type—***`Close`***.

Open/Close Switch

NI-SWITCH takes responsibility for which physical switch to open or close, so you can focus your attention on connecting signals. In other words, the operations take the two signal points (or *channels*) that you want to connect, rather than the name of a physical switch. As a result, you can not only control a simple switch card, but also handle situations where you need multiple switches to connect different channels, such as connecting two columns in a matrix.

Example 2-2 shows, first in LabVIEW and then in C code, how to connect channels on a switch card. For this example, assume you have a general-purpose relay card, such as the SCXI-1160 or SCXI-1161. Because each switch is independent of the other, you can open or close it independently of any other switch. Also, the example assumes a Form A switch which has two channels: input (*ch* for channel) and an output (*com* for the common).

Figure 2-1 illustrates Forms A, B, and C switches.

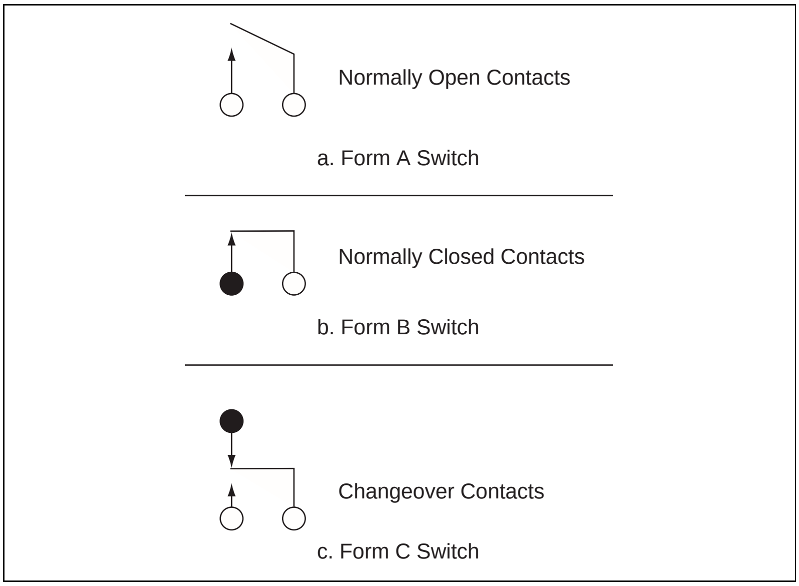
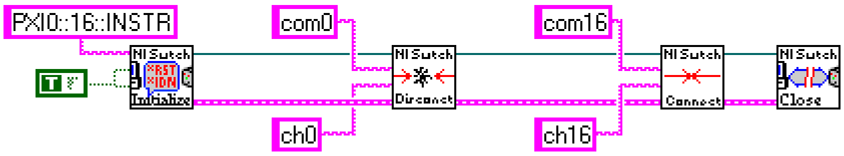


Figure 2-1. Contact Forms

Example 2-2 General Purpose Switches



Note *C code examples 2-2 through 2-5 use* `bold text` *to distinguish lines of code that are different from Example 2-1.*

```
#include "niswitch.h"

int main (void)
{
   ViSession instr;             /* Communication Channel */
   ViStatus status;             /* For checking errors */


   /* Begin by opening a communication channel to the instrument */
   status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
   if (status < VI_SUCCESS) {
     /* Error Initializing Interface...exiting */
     return -1;
   }


   /* NOTE: For simplicity, we will not show any other error checking. */


   /* Disconnect Channel 0 from the common (open the switch) */
   status = niSwitch_Disconnect(instr, "com0", "ch0");


   /* Connect Channel 16 to the common (close the switch) */
   status = niSwitch_Connect(instr, "com16", "ch16");
   /* Close communication channel */
   status = niSwitch_close(instr);
   return 0;
}
```

Example 2-2 Discussion

Example 2-2 breaks down into the following steps:

1. The program begins and ends with the same steps as described in Example 2-1. Refer to that example for more information on steps not in bold.
2. The first command is ***`Disconnect`***. This operation tells the driver to disconnect the signal connection from the *ch0* channel to the *com0* channel. In the case of the general-purpose switch, this merely opens switch 0.

Note ***There is no significance to which switch is listed first.***

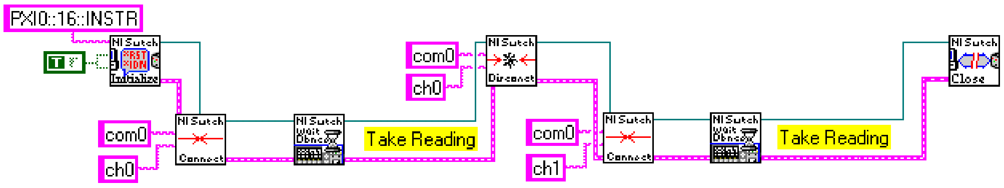
3. The second command is ***`Connect`***, which connects the channels of *com16* and *ch16*. Again, this is basically closing switch 16 for the general-purpose switch.

Manual Scanning

Example 2-3 shows how to use the instrument driver to control a multiplexer. Notice that the programming is very similar.

Caution ***Be sure to control the multiplexer so that it measures only one channel at a time unless you are switching large inductive loads. Failure to do so could result in two of the channels being shorted together.***

Example 2-3 Multiplexer



```
#include "niswitch.h"

int main (void)
{
    ViSession instr;            /* Communication Channel */
    ViStatus status;            /* For checking errors */


   /* Begin by opening a communication channel to the instrument */
   status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
   if (status < VI_SUCCESS) {
     /* Error Initializing Interface...exiting */
     return -1;
   }


   /* NOTE: For simplicity, we will not show any other error checking. */


   /* Close switch #0 */
   status = niSwitch_Connect(instr, "com0", "ch0");
   status = niSwitch_WaitForDebounce(instr, 1000);


   /* INSERT CODE TO MAKE READING */


   /* Open switch #0 */
   status = niSwitch_Disconnect(instr, "com0", "ch0");
   /* Close switch #1 */
   status = niSwitch_Connect(instr, "com0", "ch1");
   status = niSwitch_WaitForDebounce(instr, 1000);


   /* INSERT CODE TO MAKE READING */


   /* Close communication channel */
   status = niSwitch_close(instr);
   return 0;
}
```

Example 2-3 Discussion

Example 2-3 breaks down into the following steps:

1. The program begins and ends with the same steps as described in Example 2-1. Refer to that example for more information on steps not in bold.
2. The first command is ***`Connect`***. As described in Example 2-2, this operation connects the first channel to the output (common) of the multiplexer. Before you do anything else, you should wait for the switch to settle (debounce) before taking a reading. To do this, call the ***`Wait For Debounce`*** operation. In this example, the wait operation takes 1000 ms (one second) as its **timeout** parameter. Notice that this is the default value for LabVIEW, so it is not wired.
3. The next step is for the measurement device to take a reading from the output of the multiplexer.
4. After making the measurement, use ***`Disconnect`*** to break the connection to channel 0 and ***`Connect`*** to make the new connection to channel 1. Notice, however, that the disconnect operation was not followed by a ***`Wait For Debounce`*** operation. Opening channel 0 before closing channel 1 ensures that the two channels do not short together, as long as they are on the same card. However, since your concern is only that the board settles after channel 0 opens and channel 1 closes, you do not need to waste time waiting for channel 0 to settle. Therefore, you open channel 0 and then immediately close channel 1. In electromagnetic relays, settling times often are measured in several milliseconds. By not waiting for debounce during the open stage, you can significantly reduce the time it takes to perform a scan.

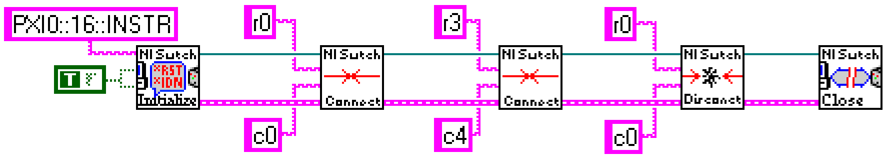
Warning ***If you are using multiple models of switch cards wired together, remember to wait for debounce when opening the channels. If you do not explicitly wait for debounce, differences in the switch times may cause electrical shorts.***

5. Now you can take another measurement, this time measuring the signal on channel 1. This cycle can continue indefinitely.

Matrix Operations

The last example under manual control of the switch is the *matrix*. The NI-SWITCH API uses special channel strings for the matrix but the general operation is the same. This is because the API focuses on creating connections rather than opening and closing switches.

Example 2-4 Matrix



```
#include "niswitch.h"

int main (void)
{
    ViSession instr;            /* Communication Channel */
    ViStatus status;            /* For checking errors */


   /* Begin by opening a communication channel to the instrument */
   status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
   if (status < VI_SUCCESS) {
     /* Error Initializing Interface...exiting */
     return -1;
   }
   /* NOTE: For simplicity, we will not show any other error checking. */


   /* Connect the Matrix Point (row=0, col=0) */
   status = niSwitch_Connect(instr, "r0", "c0");


   /* Connect the Matrix Point (row=3, col=4) */
   status = niSwitch_Connect(instr, "r3", "c4");


   /* Disconnect the Matrix Point (row=0, col=0) */
   status = niSwitch_Disconnect(instr, "r0", "c0");



   /* Close communication channel */
   status = niSwitch_close(instr);
   return 0;
}
```

Example 2-4 Discussion

Example 2-4 breaks down into the following steps:

1. The program begins and ends with the same steps as described in Example 2-1. Refer to that example for more information on steps not in bold.
2. The first command is ***`Connect`***. This operation works the same for controlling switches on general-purpose and multiplexers, with the exception that the default names are different.
3. You then continue to open and close points on the matrix. This cycle can continue indefinitely.

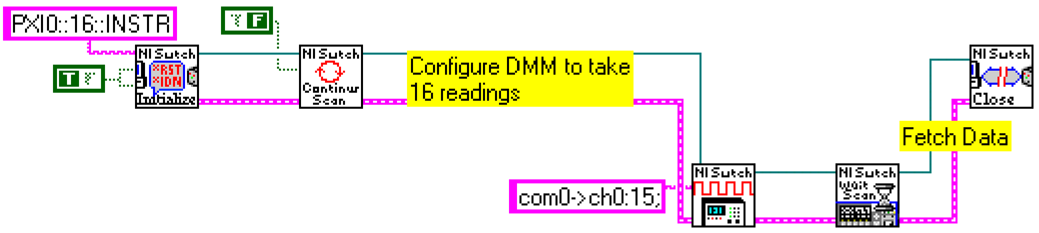
Basic Scan

When the number of channels to scan becomes large, any improvement in efficiency can have a dramatic effect on the overall scan time. One way to improve efficiency is to have the measurement device and the switch talk to each other to make sure each runs as fast as possible. This is where scanning comes in. Using scanning, you download the set of switches to open and close into the memory of the card. The scanning process then uses triggers to communicate between the measurement (or source) device and the switch card itself.

Using scanning is very straightforward. Example 2-5 shows how to scan 16 channels with a scanning DMM. For more details of the scan list syntax itself, refer to the *Scan List Syntax* section in Chapter 5, *Scanning*.

Note ***What the switch considers*** *Trigger Input* ***is often called*** *Volt Meter Complete* ***by the DMM.***

Example 2-5 Scanning



```
#include "niswitch.h"

int main (void)
{
    ViSession instr;             /* Communication Channel */
    ViStatus status;             /* For checking errors */


   /* Begin by opening a communication channel to the instrument */
   status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
   if (status < VI_SUCCESS) {
     /* Error Initializing Interface...exiting */
     return -1;
   }


   /* NOTE: For simplicity, we will not show any other error checking. */

   /* Turn off Continuous mode. We want a one-shot scan */
   status = niSwitch_SetContinuousScan(instr, VI_FALSE);
```

```
    /* CONFIGURE THE DMM TO TAKE 16 READINGS AND WAIT FOR A */
    /* TRIGGER BEFORE STARTING EACH READING. ALSO */
    /* ASSERT A TRIGGER AFTER EACH READING. */



    /* Now scan... */
    status = niSwitch_Scan(instr, "com0->ch0:15;");


    /* Wait for Scan to complete */
    status = niSwitch_WaitForScanComplete(instr, 5000);


    /* DOWNLOAD DATA FROM DMM */


    /* Close communication channel */
    status = niSwitch_close(instr);
    return 0;
}
```

Example 2-5 Discussion

We can break Example 2-5 down into the following steps.

1. The program begins with the same steps as described in Example 2-1. Refer to that example for more information on steps not in bold.
2. At this point, you set the continuous mode to *False*. This indicates whether the switch card should cycle through the scan list or stop at the end of the scan list. In this case, you want the scan to stop after 16 channels. However, if you wanted to read the same channels multiple times, you would set up the 16 channels in the scan list and set the continuous mode to *True*.
3. Now that the switch is configured, you need to configure the measurement/source instrument. In this example we are assuming a scanning DMM. We cannot show the code necessary because that is dependent on the DMM. However, we can describe the set of steps necessary to configure the DMM. First you configure the DMM to take readings only when it detects a trigger on its trigger input. Next you configure it to generate triggers after it takes a reading. Finally, you tell the DMM to take 16 measurements.

4. You can now initiate the scan. The NI-SWITCH API has several operations for performing a scan, but the simplest one is the ***`Scan`*** operation. Here you prepare a list of which channels to scan and in what order. The operation automatically programs the switch card and closes the first entry in the scan list. Because the switch card is configured to generate a trigger when a switch is closed, the first switch kicks off the handshaking. Refer to Chapter 5, *Scanning*, for more information about scanning operations and the scan list syntax.
5. The ***`Scan`*** operation automatically returns when the scanning mode is enabled. To determine when the scan is complete, call ***`Wait For Scan Complete`***. Notice that you can use this operation only when the switch is *not* in continuous mode. If you are using continuous mode, check the status of the other instrument to see when the scan is complete.
6. The last step is to retrieve the measurements from the DMM when the scanning is complete.

3 Getting Started

This chapter contains an overview of the NI-SWITCH Application Programming Interface (API), defines special terms you need to understand, and introduces the various operations defined by the VXI*plug&play* specifications on instrument drivers. The operations specified by these documents are standard across all VXI*plug&play* instrument drivers. In addition, this chapter presents information on other operations unique to the NI-SWITCH driver, and gives an overview of the main groups of attributes.

Before reading this chapter, you should have reviewed Chapter 2, *Introductory Programming Examples*, to become familiar with the flow of programs using the NI-SWITCH driver.

Refer to the online help for a complete list of the operations and their parameters.

Introduction

The NI-SWITCH API is designed to be consistent with the VXI*plug&play* alliance, VISA, and IVI technology. Because of these technologies, there are several object-oriented concepts that exist in the API. When you consider a description of an object, such as a car, computer, or switch card, it has two main groups of information associated with it.

- The first group is *attributes*. These give state information about the object. For example, attributes of a car include its color, the number of doors, and whether it is currently running. For a switch card, attributes include the number of channels, the bandwidth of the card, and whether the switches on the card are debounced. In addition, attributes can also control features of the switch, such as which trigger lines the switch card should use.
- The next group of information is the object's *operations*. These are often called the *verbs of the object* because they describe what the object can do. For example, a car's operations include accelerating, braking, and turning. For a switch card, operations include opening or closing a switch.

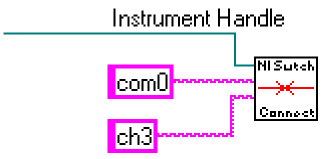
- In addition to these two groups, another feature of objects is *sessions*, or *handles*. These are the unique identifiers of the object and are used to communicate with the object. They are similar to items such as file I/O handles. Sessions, attributes, and operations are described in more detail in the following sections.

Note ***You can use a switch card to route signals for measurements or for sourcing. You can also use it to control a circuit by making or breaking an electrical connection. However, throughout this manual, when describing switching applications, we commonly refer to the switch card being connected to a measurement device and the switch card passing signals through to the measurement device. This is not meant to restrict the use of the switch card, but rather to simplify the documentation. At almost any point where we discuss a measurement device, you can apply the same information to the other possible applications of the switch card.***

Naming and Terminology

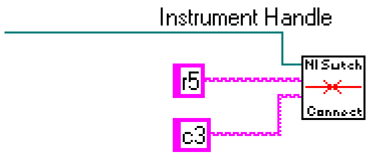
A switch module consists of a series of switches, either interconnected as in the case of a matrix or multiplexer, or independent as in the case of the general-purpose switch. However, from the user's point of view, the switch is no more than a way to connect signal paths. If we adapt this point of view to the switch module, it is no more than a black box with a variety of signal connections, or *channels*. It is in this light that National Instruments designed the NI-SWITCH driver.

Consider the case of a simple multiplexer. This switch module consists of a variety of channels that are multiplexed to a single channel, called the *common*. Therefore, if you want the switch module to route the signal on the input channel—although the definitions of *input* and *output* can be reversed on most switch modules—to the common, you tell the driver to connect the two channels, as follows:



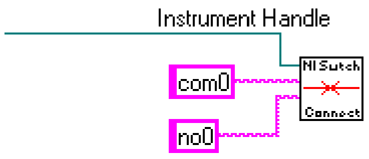
```
status = niSwitch_Connect(instr, "com0", "ch3");
```

In the case of a matrix, the terms *channel* and *common* are typically replaced with the terms *row* and *column*. However, the driver still considers them as channels. Notice the similarity in the following code to connect row 5 to column 3:



```
status = niSwitch_Connect(instr, "r5", "c3");
```

Finally, in the case of a general-purpose switch module, the independent switch can be considered a multiplexer in its own right. For example, you can consider a form A switch as a 1 x 1 multiplexer, and a 1-form C switch as a 2x1 multiplexer. In the case of a 1-form C switch, its channels are often called COM (common), NC (normally closed) and NO (normally open). Again, we can use the connect operation:

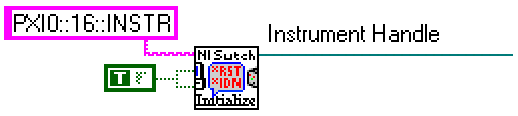


```
status = niSwitch_Connect(instr, "com0", "no0");
```

Session Communication

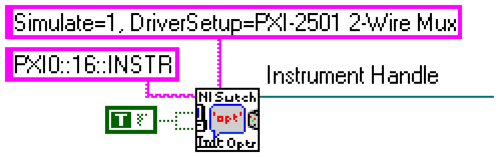
The NI-SWITCH driver is a *scalable* driver, which means it can talk to any of the National Instruments switches. Therefore, when you want to use the driver, you must be able to uniquely identify the appropriate hardware. You do this through the ***Initialize*** operation, where you pass in a unique descriptor of the hardware. This descriptor gives the driver the physical address of the hardware. The driver then returns a special handle, called a session, to you. Whenever you want to perform any action on this hardware, you pass the session back to the driver.

For example:



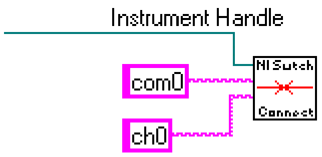
```
status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, &instr);
```

This code opens communication with a PXI switch card at address 16. The session is returned in the last parameter, in this case in the variable **instr**. As an alternative to the ***Initialize*** operation, consider a similar operation—***Initialize With Options***. This operation initializes the driver into a state you specify using options you pass into the operation. For example, you can operate the NI-SWITCH driver in simulation mode, in which you can run programs without needing the switch hardware actually connected to the computer. To open a session to the simulation driver, you would use ***Initialize With Options*** as follows:



```
status = niSwitch_InitWithOptions("PXI::10::INSTR", VI_TRUE, VI_TRUE,
                   "Simulate=1, DriverSetup=PXI-2501 2-Wire Mux", &instr);
```

To talk to the hardware, use this session as the first parameter of every operation from then on, as shown below:



```
status = niSwitch_Connect(instr, "com0", "ch0");
```

This operation connects channel 0 to the common on the hardware that **instr** points to. When the communication is complete, close the session by using the ***Close*** operation.

☞ **Note** ***You can have only one session open to a unique piece of hardware at a time.***

Due to the advanced features of IVI, such as state caching, you should not have multiple sessions to—or multiple views of—the same hardware. Therefore, if your application is multi-threaded, each thread shares the same session. For protection between the threads, NI-SWITCH includes a set of lock operations.

Attributes

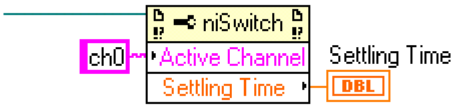
You can use attributes to get information about the state of the card, or to set the state of the card. For example, by retrieving an attribute from the driver, you can determine whether the switches on the card have debounced, or you can set the source of a trigger. Some NI-SWITCH *helper* operations even let you utilize attributes without accessing them directly. For example, the ***Is Debounced*** operation also tells you whether the switches on the card have settled, and ***Configure Scan Trigger*** sets up the necessary trigger parameters in a single call.

Accessing Attributes

Accessing attributes in LabVIEW works slightly differently than for C and Visual Basic. LabVIEW uses a feature known as the *property node*. A LabVIEW programmer can use the property node to get and set multiple attributes at the same time. You can access the attributes in LabVIEW only through the property nodes, which display the list of possible attributes you can access.

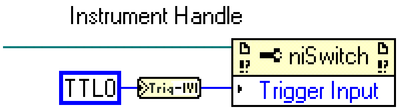
C and Visual Basic users use special functions, such as `niSwitch_GetAttributeViInt32()` and `niSwitch_SetAttributeViBoolean()`. In these languages, the attributes themselves are identified by their names, which always start with *`NISWITCH_ATTR_`*.

The following examples show how to get and set attributes in LabVIEW and C:



```
status = niSwitch_GetAttributeViInt32(instr, "ch0",
                                      NISWITCH_ATTR_SETTLING_TIME, &attrVal);
```

or



```
status = niSwitch_SetAttributeViInt32(instr, "", NISWITCH_ATTR_TRIG_INPUT,
                                      NISWITCH_VAL_TTL0);
```

These two operations also point out two other special features of the attribute operations. First, notice that the C operations take the data type of the attribute as part of the name. Therefore, there is a unique function for each data type (Boolean, integer, and so on).

The next thing to notice about the operations—for both LabVIEW and C—is the **channel** parameter. NI-SWITCH attributes can be either global to the entire board, or local to each channel of the board. Therefore, you must indicate which channel, if necessary, you want to communicate with.

In LabVIEW, this is done by setting the ***`Active Channel`*** Attribute. All channel-based attributes below it are then directed to that channel.

Attribute Functionality

The NI-SWITCH online help fully describes all of the attributes for the NI-SWITCH driver. However, review this section for a general overview of the main groups so that you can better understand what information and control is available through the attributes.

Read-Only State Attributes

The first group of attributes is the read-only state attributes. A programmer can use these attributes to query the switch card for basic information about the switch. This makes it possible for a program to be scalable across different cards by being able to configure itself depending on the actual hardware present (such as number of channels).

The following attributes are *global* across all channels:

- ***`Number of Rows`***
- ***`Number of Columns`***
- ***`Wiring Mode`***

The other attributes in this group contain the specifications for the switches, such as the ***`Bandwidth`*** Attribute and the ***`Maximum AC Voltage`*** Attribute. Notice that this information is specific to the switch mentioned in the channel name of the get and set attribute operations. Therefore, getting the bandwidth value for the first switch in a multiplexer will not tell you the bandwidth through the card, but only through that specific switch.

Operation Attributes

Another group of attributes controls basic operation of the switch card. For example, the ***`Continuous Scan`*** Attribute controls whether the scanning continuously loops through the scan list, or stops at the end of the scan list. Other examples for the scan list are the ***`Scan List`*** Attribute, which contains the actual scan list, and the ***`Scan Mode`*** Attribute, which controls the break-before-make and break-after-make functionality. In this same group are several attributes that control triggering. These attributes, such as the ***`Trigger Input`*** Attribute and the ***`Scan Advanced Output`*** Attribute, are described more thoroughly in Chapter 5, *Scanning*.

Operations

An operation is another name for a VI in LabVIEW or a function in C or Visual Basic. These operations give you full access to the NI-SWITCH driver, including access to attributes (through the get and set operations) for C and Visual Basic users. Remember, in LabVIEW you use attribute nodes to access attributes.

Consult the online help for more information on NI-SWITCH operations.

Overview of VXI*plug&play* Required Operations

Initialize and Close

As described earlier in this chapter, the central component to communicating with the driver and hardware is the session. Use ***`Initialize`*** and ***`Close`*** in your program to create and destroy the session, respectively. In addition, ***`Initialize`*** can require a verification that the address actually corresponds to the correct hardware, and can perform a complete reset on the board.

Reset

In the ***`Initialize`*** operation, the program can order a board reset. However, you can achieve the same functionality through the ***`Reset`*** operation. In both cases, the reset causes the board to return to its powered-on state.

Remember that the definition of this state can vary between latching and non-latching relay boards. For the non-latching boards, the power-on state is always the same, with the relays returning to their *Normally Closed* (NC) position. However, latching relays can maintain their state between power cycles, depending on the design of the board. However, all latching boards, when reset, return to their reset position.

Self Test

The ***`Self Test`*** operation is designed to verify that the board is operational. The level of this verification depends on what information the driver can obtain about the board. In general, the NI-SWITCH driver can verify that the board is responding and the driver can access the registers of the hardware.

Error Query and Error Message

All operations return status information that indicates whether the operation executed successfully. However, you can use the ***`Error Query`*** operation to retrieve the status code returned by the last operation called. The ***`Error Message`*** operation translates an NI-SWITCH status code into human-readable text, which can be displayed via a pop-up to the user.

Revision Query

`Revision Query` returns the revision of the instrument driver, in this case the revision of NI-SWITCH, as well as the firmware revision.

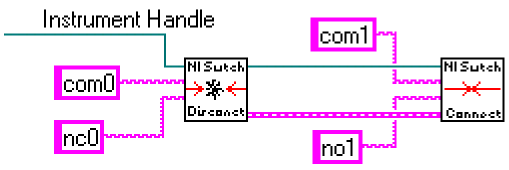
4 Manual Switch Control

This chapter describes operations you can use to control a switch card manually—rather than automatically through scanning operations—and discusses the effect of switch topology on manual operations. This chapter also summarizes the reset functions and the various possible levels of reset.

Open and Close

General-Purpose Switch Topologies

The general-purpose topology refers to a switch card containing multiple switches that are completely independent of one another. Examples of general-purpose switch cards are the SCXI-1160 16-Channel Switch and the SCXI-1161 8-Channel, High-Power Switch. These National Instruments cards are designed to make or break electrical circuits, much as a light switch in your house controls the power to the light bulb. Because each of these switches is independent, the general-purpose switch is one of the simplest switch cards to use. Below are LabVIEW and C examples of opening and closing switches on a general-purpose switch card.



```
/* Open switch #0 */
status = niSwitch_Disconnect(instr, "com0", "nc0");
/* Close switch #1 */
status = niSwitch_Connect(instr, "com1", "no1");
```

The parameters are the session and the channel names. The session parameter is the communication handle returned by the ***Initialize*** operation, which the program must call before issuing any other operation to the hardware. The channel name parameters indicate which of the independent switches are to be activated.

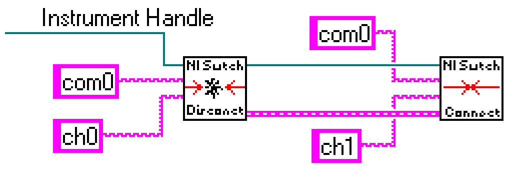
Note ***All National Instruments switch cards number their channels starting from zero. Refer to the hardware information contained in the switch card's user manual for a list of the channel numbers for the specific switch card.***

Multiplexers, Scanners, and Trees

Multiplexers, scanners, and trees can all expand the number of channels available to a measurement or source device. While they have subtle differences, for the purpose of the software, they can be considered the same and will be documented as such. Whenever you see the word *multiplexer*, you can assume it is interchangeable with *scanner* or *tree*, unless otherwise noted.

The output from a multiplexer can vary depending on the configuration of the card. For example, the NI 2503 is a 24-channel, two-wire multiplexer. This means that it can select any one of 24 differential signals to pass through to the output. However, the output could be either the main output signals or the analog bus. You could also configure the NI 2503 as two separate 12-channel, two-wire multiplexers. In this case, the outputs are different from those in the single multiplexer case.

Programming the multiplexer, however, looks the same as for the general-purpose topology. For example:



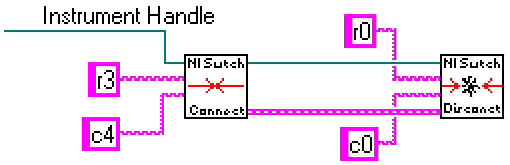
```
/* Open switch #0 */
status = niSwitch_Disconnect(instr, "com0", "ch0");
/* Close switch #1 */
status = niSwitch_Connect(instr, "com0", "ch1");
```

The parameters are the session and the channel names. The session parameter is the communication handle returned by the **`Initialize`** operation, which the program must call before issuing any other operation to the hardware. The channel name parameters indicate which channel(s) to activate.

Matrices

The final topology covered in this manual is the *matrix*. The matrix allows you to connect any input to any output. A typical use for a matrix is to have one side—for example, the columns—connected to different instruments and the other side—for example, the rows—connected to the signal points. For example, a 64 x 4 matrix could have a DMM, oscilloscope, function generator, and power supply connected to the four columns and have 64 signal points to which any one (or multiple) of the instruments could connect.

Because the point of a matrix is to make a connection between a row and a column, the output of the matrix is no longer implicit. For this reason, new channel names are required to handle matrices. Below are two examples of the operations taken from Chapter 2, *Introductory Programming Examples*.



```
/* Close the Matrix Point (row=3, col=4) */
status = niSwitch_Connect(instr, "r3", "c4");
/* Open the Matrix Point (0, 0) */
status = niSwitch_Disconnect(instr, "r0", "c0");
```

The parameters are the session and the row-and-column names. The session parameter is the communication handle returned by the ***Initialize*** operation, which must be called before any other operation to the hardware. The row and column names indicate which connections to make.

Reset

Chapter 3, *Getting Started*, discussed two ways to reset the switch card to its powered-on state. These are through the reset parameter in ***Initialize***, or explicitly through a call to ***Reset***. However, there is another operation that provides more granularity to reset. You can use the

Disconnect All operation to disconnect all existing paths. This operation disconnects the paths without affecting the configuration information already stored by the switch module.

Analog Bus

In addition to the common connections, switches can have an output that has its own switch—the analog bus. This output is designed so that switch modules can share a common wire back to the instrument. To prevent shorting signals, you can have only one switch module on the analog bus at a time—that is, its analog bus switch is closed.

Notice that because the analog bus has a switch, it appears as a channel connected to the common, as illustrated in Figure 4-1.

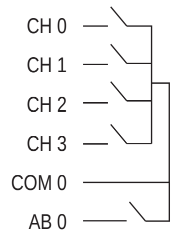
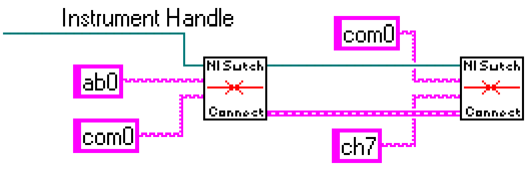


Figure 4-1. Analog Bus Connected to Common

For this reason, connecting to the analog bus can be a two-step process:



1. Connect the analog bus to the common.

   `niSwitch_Connect(instr, "ab0", "com0");`

2. Connect the channels to the common.

   `niSwitch_Connect(instr, "com0", "ch7");`

The reason you cannot simply connect *ab0* and *ch7* with one call to ***`Connect`*** is that it would require the use of another channel. The ***`Connect`*** operation is capable of closing multiple switches so that it can make the path, but it cannot do this if it means going through extra channel switches.

However, there is a way to allow the NI-SWITCH driver to perform the *ab0*-to-*ch7* connection in one step, which is by telling the driver that *com0* is no longer a channel, but instead a resource available to the driver to make connections.

To convert a channel into a resource, set the ***`Is Configuration Channel`*** Attribute to *True*. After you mark a channel as a configuration channel, the driver can use it to create paths. Notice, however, that no operations will work with that channel name except the getting and setting of the ***`Is Configuration Channel`*** Attribute.

5 Scanning

This chapter discusses the basic building blocks of scanning, such as scan lists and trigger configuration. In addition, this chapter includes examples of how to do scanning with either the internal software scanning architecture or with hardware scanning.

Overview

In many cases, you can control the switches on a switch card by supplying a list of switches to scan. Not only multiplexers, but also general-purpose and matrix cards can use scanning to sequence through a set of patterns. In these cases, the scanning may actually involve software internal to the driver, and you can realize significant speed advances due to the asynchronous architecture of the driver. In the cases with multiplexers and some matrices, however, you can download the scan list itself directly to the hardware and execute it independently of the controller.

☞ **Note** ***Whenever the switch card is scanning—the `Is Scanning` operation returns True—the only operations you can use are `Is Scanning`, `Wait For Scan Complete`, `Abort Scan`, and the various `Get Attribute` operations. All other operations will return an error if executed during scanning.***

Scan List Syntax

The first step in scanning is to tell the driver what channels to scan and in what order. You do this by preparing a *scan list string*, which is then parsed by the driver. The syntax for the string is fairly simple and straightforward. We will start by giving some basic examples of scan strings and then list the exact syntax.

Basic Scan

The first example is to sequentially scan 32 channels. Note that whether to repeat the scan list is not specified here, but in the **`Set Continuous Scan`** operation. The colon in the string indicates the range. The semi-colon at the end indicates that the scan should wait for the final trigger from the measurement device before the scan is considered complete or before starting the scan string over again:

```
com0->ch0:31;
```

If you want to scan channels in a random order, separate the individual channels with a semi-colon:

```
com0->ch0; com0->ch16; com0->ch31; com0->ch8;
```

You can also combine random and sequential scanning, as in:

```
com0->ch0:8; com0->ch16:24;
```

In some cases, you may want to close multiple switches in a single trigger event. This is accomplished through the ampersand (&). For example, if you have configured the NI 2503 as two 12x1 multiplexers, you can scan:

```
com0->ch0 & com2->ch12; com0->ch1 & com2->ch13;
```

Remember that when scanning, there is the question of what to do with the previous connections as you proceed with the scan. What happens to channel 0 when channel 1 is to close (as in the first scan example above) depends on the **Scan Mode** parameter of ***`Configure Scan List`*** (or the setting of the ***`Scan Mode`*** Attribute, which is set automatically by the ***`Configure Scan List`*** operation):

- If the value is set to ***`BREAK BEFORE MAKE`***, channel 0 opens before channel 1 closes.
- If the value is ***`BREAK AFTER MAKE`***, channel 0 opens after channel 1 closes. The break-after-make is useful when dealing with currents and inductive loads that could be damaged by sudden breaks in current flow.
- The final possible value is ***`NONE`***. This means that the user takes over all responsibility of breaking previous connections and the driver should take no implicit action on behalf of the program. This is useful when scanning through patterns with general-purpose or matrix cards. To manually break a previously made connection, use the tilde (~) symbol, as shown below:

  ```
  r0->c0; ~r0->c0 && r2->c2;
  ```

Notice also the use of the && (double ampersand). Like the single ampersand described previously, it indicates that both `"~r0->c0"` and `"r2->c2"` are to occur before the scan advanced trigger is sent. However, by placing a double ampersand, the string tells the driver to wait until the `"~r0->c0"` disconnection has settled before connecting `"r2->c2"`.

Ending Semi-colon

You may have noticed the ending semi-colon on the various scan strings. This syntax is very important because it tells the switch to wait for the final trigger before opening the last switch again. For example:

```
com0->ch0; com0->ch1
```

This scan string would leave channel 1 closed at the end of the scan and would cause the next scan to fail—the scan would close channel 0 while channel 1 was still closed.

Notice for the case of Break After Make, you should have the scan engine return to the start state. For example:

```
com0->ch0; com0->ch1; com0->ch0
```

Notice the final "`;`" is missing. This is because we want the `com0->ch0` path to remain when the scan is over. Failure to do this would result in no switches closed, which goes against the purpose of Break After Make.

Breakpoints

Another feature you can use is inserting breakpoints into the scan list. The breakpoint halts the scan, which can be detected by the ***`Is At Breakpoint`*** operation. The user can be notified when this occurs and take action. This action could be part of a debugging process, or a time to notify the user to take some action (such as manually change a setting on the device under test). To specify a breakpoint, use the less-than (<) and greater-than (>) characters to identify the reserved word `BREAK`. For example:

```
com0->ch0:8; <BREAK>; com0->ch9:15;
```

Counting Triggers

Another example of a reserved word is `<REPEAT n>`. You can insert this word in your scan list so that you do not have to type in multiple, identical entries. For example, if two cards are scanning together, they each need a scan list and each needs a section in the scan list to count triggers while the other card is taking action. For example, the following two scan lists do exactly the same thing:

```
com0->ch5; com0->ch6; ; ; ; com0->ch7;
com0->ch5; com0->ch6; <REPEAT 3>; com0->ch7;
```

In the first example, the scanner selects channel 5, waits for a trigger, selects channel 6 and then counts four triggers before selecting channel 7.

The second example is shorthand for describing this same scan. The `REPEAT` command takes whatever is in the same section as the command word and repeats it *n* number of times, including the ending semi-colon. You can find more information on this subject in the *Trigger Configuration* section in Appendix B, *Multiple Card Scanning*.

Tables 5-1 and 5-2 summarize the various scan syntax rules and reserved words, respectively.

Scan Delay

The switch card generates a scan advanced trigger after all the switches have settled. This alerts the measurement device that it can now take a measurement. The switch cards themselves are designed to wait a specified amount of time (see the description of the ***`Settling Time`*** Attribute in the NI-Switch online help) to ensure the switch has settled. However, due to capacitance in the system, it may take more time for the switch to settle. To increase the time between the settling time and the scan advanced trigger, you can set the **Scan Delay** parameter in the ***`Configure Scan List`*** operation.

Scan Advanced Trigger

A scan advanced trigger is generated once for each set of path creations done in between semi-colons. For example:

```
com0->ch0; com0->ch1;
```

This generates a scan advanced trigger after connecting ch0 and again after connecting ch1. Even if there are multiple commands, there is still only one scan advanced per set of connections. For example, the same set of triggers will appear for the following example:

```
com0->ch0 && com2->ch16; com0->ch1 && com2->ch17
```

Table 5-1. Scan Syntax Descriptions

Character	Example	Description
“:” Colon	`com0->ch0:31;`	*Range*. This character tells the driver to scan all the channels from the one specified to the left of the colon to the one specified to the right.

“;”
Semi-colon
	`com0->ch0;`
	`com0->ch1;`
		Wait-For-Trigger. This character tells the driver to wait for an input trigger. In this example, the driver waits for a trigger after closing channel 0. When the trigger occurs, it then opens channel 0 and closes channel 1.

“&”
Ampersand
	`com0->ch0 &`
	`com1->ch8;`
		List. This character tells the driver to connect all the paths separated by the ampersand at the same time, that is, before the next trigger event. These actions will complete before the scan advanced output trigger is asserted. Notice that there is no ordering guaranteed by the driver, except that all channels will have settled by the time the semi-colon is reached.

“~” Tilde	`~com0->ch0;`	*Break connection*. This character tells the driver to disconnect the path. Notice that the scan advanced trigger is not generated by disconnecting a path. Only path connections can generate a scan advanced trigger.
“<>” Reserved Word	`<BREAK>`	Used for *reserved words*, such as the breakpoint command, shown to the left.

“&&”
Double Ampersand
	`com0->ch0 &&`
	`com1->ch8;`
		Enforce ordering. When a single ampersand is used, it tells the driver the operations are to be done and settled before the scan advanced trigger is sent. The double ampersand tells the driver to have the operation to the left of the double ampersand settled before proceeding to the next operation in the scan. Notice that there is no trigger activity due to the &&.

Table 5-2. Reserved Word Descriptions

Reserved Word	Example	Description
BREAK	`com0->ch0:8 && <BREAK>; com0->ch9:15;`	*Breakpoint*. This specifies to halt the scan and interrupt the driver. This can be detected through the ***`Is At BreakPoint`*** operation. When you are ready to resume scanning, execute the ***`Continue From BreakPoint`*** operation. Notice that if the breakpoint should occur after switch closures but before the scanner advanced, the break command should follow the channel command as shown in the example (using the &&). Otherwise, put the break before the channel commands.
REPEAT n	`com0->ch8; <REPEAT 4>; com0->ch16;`	*Repeat*. Repeats the action in the section where the REPEAT command word is found *n* times. This is typically used to wait for multiple triggers while another switch card is scanning, but you can also use it to scan the same channel multiple times. For more information on this topic, refer to the *Counting Triggers* section, earlier in this chapter.

Triggering

Configuring the triggering options is very simple for the majority of scanning applications. However, the NI-SWITCH driver does offer a wide variety of triggering options to cover many of the side cases. To keep the descriptions simple, this section describes a common case. For details of more advanced triggering options, refer to Appendix B, *Multiple Card Scanning*.

There are two common cases when doing triggering. The first case, as shown in Figure 5-1, is hardware scanning where the measurement device and the switch handshake between each other. In this case, configuring the triggers is as simple as setting trigger values in ***`Configure Scan Trigger`***. These values can be either ***`EXTERNAL`***—which typically is either the front or rear trigger connector—or one of the TTL or ECL trigger lines.

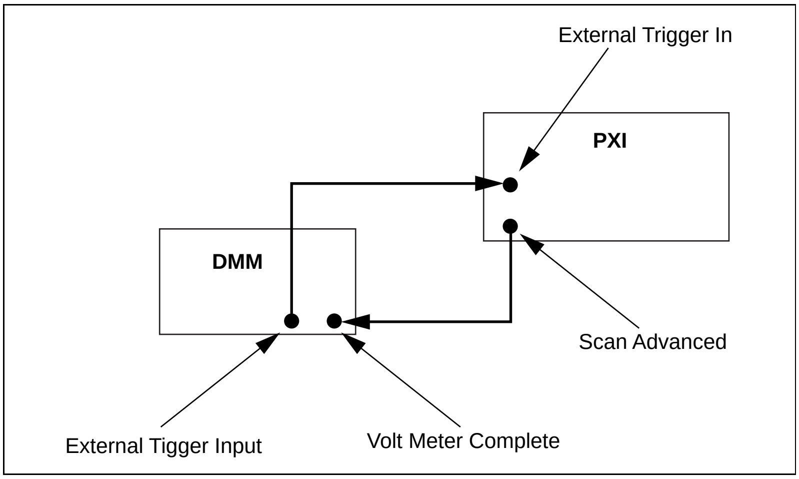


Figure 5-1. Simple Two-Wire Handshake with DMM

Another value for the **trigger** parameter is ***`Software Trigger Function`***. This setting tells the switch card that the trigger will not come via a hardware trigger line, but rather from a software command. The program sends the command via the ***`Send Software Trigger`*** operation. This is useful if the timing information is more complicated than a simple trigger line and requires the controller to calculate when to sequence the scan.

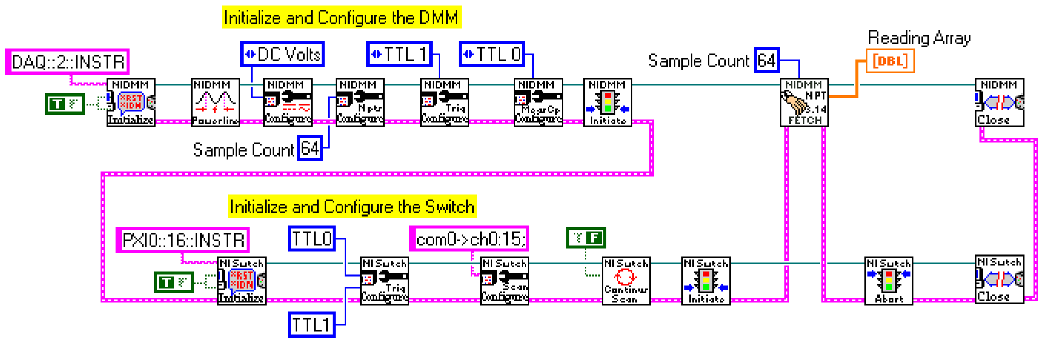
Scan Operations

In Example 2-5, *Scanning*, we saw that performing a scan takes very little in terms of coding. The ***`Scan`*** operation uses the scan list as its main parameter, parses the scan list, programs the hardware, and initiates the scan before returning. Notice that it also does not wait for the scan to complete before it returns.

You can use the ***`Is Scanning`*** or ***`Wait For Scan Complete`*** operations to determine if the card is currently in scan mode. However, remember that if the continuous mode (via ***`Set Continuous Mode`***) is set to *True*, the switch card has no way of knowing the scan is complete, so neither of these operations will tell you very much. The preferred tactic is to track the state of the scan by monitoring the measurement device, since it will have the actual count of measurements to be made.

Example 5-1 Scan Programming Example

The following example shows how to use the NI 4060 DMM and the NI 2503 24-channel multiplexer together to perform a hardware scan when the switch is set to a continuous scan mode.



```
#include "nidmm.h"
#include "niswitch.h"

int main (void)
{
    ViSession DMMinstr;           /* Communication Channel */
    ViSession SWITCHinstr;        /* Communication Channel */

    /* DMM Variables */
    ViInt32 maximumTime = 5000;   /* Number of ms to wait for operation */
    ViReal64 sampleInterval = 0.0;/* Time waited after trigger */
    ViInt32 sampleCount = 1;      /* Number of samples per trigger */
    ViInt32 trigCount = 64;       /* Number of triggers to count */
    ViReal64 trigDelay = 0.0;     /* Time to wait from trigger */
    ViReal64 ACminFreq = 20.0;    /* Minimum Frequency of Interest */
    ViReal64 ACmaxFreq = 20000.0; /* Maximum Frequency of Interest */
    ViInt32 arraySize = 64;       /* Size of data array */
    ViReal64 arrayData[64];       /* Data from DMM */
    ViInt32 actualNumPoints;      /* Number of points returned from DMM */

    ViStatus status;              /* For checking errors */
```

```
/* Begin by opening a communication channel to the instrument */
status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE,
                        &SWITCHinstr);
if (status < VI_SUCCESS) {
    /* Error Initializing Interface...exiting */
    return -1;
}

status = niDMM_init("DAQ::2::INSTR", VI_TRUE, VI_TRUE, &DMMinstr);
if (status < VI_SUCCESS) {
    /* Error Initializing Interface...exiting */
    niSwitch_close(SWITCHinstr);
    return -1;
}


/* NOTE: For simplicity, we will not show any other error checking. */


/* CONFIGURE THE DMM TO TAKE 64 READINGS AND WAIT FOR A */
/* TRIGGER ON TTL1 BEFORE STARTING EACH READING. ALSO */
/* ASSERT A TRIGGER ON TLL0 AFTER EACH READING. */
status = niDMM_SetPowerLineFrequency(DMMinstr, NIDMM_VAL_60_HERTZ);

status = niDMM_Configure(DMMinstr, NIDMM_VAL_DC_VOLTS,
                          NIDMM_VAL_AUTO_RANGE_ON, NIDMM_VAL_5_5_DIGITS,
                          ACminFreq, ACmaxFreq);

status = niDMM_ConfigureMultiPoint(DMMinstr, trigCount, sampleCount,
                                   NIDMM_VAL_IMMEDIATE, sampleInterval);

status = niDMM_ConfigureTrigger(DMMinstr, NIDMM_VAL_TTL1, trigDelay);

status = niDMM_ConfigureMeasurementComplete(DMMinstr, NIDMM_VAL_TTL0,
                                           NIDMM_VAL_POS);

status = niDMM_Initiate(DMMinstr);



/* Start by configuring the handshake lines */
/* In this case, we are using TTL0 and 1 for triggers */
/* for the PXI switches */
```

```
   status = niSwitch_ConfigureScanTrigger(SWITCHinstr, 0.0,
                                          NISWITCH_VAL_TTL0,
                                          NISWITCH_VAL_TTL1);


   /* Provide the list of channels to scan and also say */
   /* what scan mode to use */
   status = niSwitch_ConfigureScanList(SWITCHinstr, "com0->ch0:15;",
                                   NISWITCH_VAL_BREAK_BEFORE_MAKE);


   /* Set the Continuous mode */
   status = niSwitch_SetContinuousScan(SWITCHinstr, VI_TRUE);


   /* Now scan 16 channels on the switch 4 times (16x4 = 64)... */
   status = niSwitch_InitiateScan(SWITCHinstr);



   /* DMM DETECTION OF COMPLETE SCAN AND RETRIEVE DATA */
   status = niDMM_FetchMultiPoint (DMMinstr, maximumTime, arraySize,
                                   arrayData, &actualNumPoints);



   /* Stop the Switch from scanning */
   status = niSwitch_AbortScan(SWITCHinstr);



   /* Close communication channel */
   status = niSwitch_close(SWITCHinstr);
   status = niDMM_close(DMMinstr);

   return 0;
}
```

A

Microsoft Visual Basic Examples

This appendix shows the Visual Basic syntax of the ANSI C examples given earlier in this manual. The examples use the same numbering sequence for easy reference.

Example 2-1

```
Private Sub vbMain()
        Dim instr as ViSession         'Communication Channel
        Dim status as ViStatus         'For checking errors
        Dim firmRev as Vichar * 256    'Strings for revision info
        Dim driverRev as Vichar * 256

        Rem Begin by opening a communication channel to the instrument
        status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, instr)
        if (status < VI_SUCCESS) Then
               Rem Error initializing interface ... exiting
               Exit Sub
        End If

        REM NOTE: For simplicity we will not show any other error checking

        REM Get the revision of the driver
        status = niSwitch_revision_query(instr, driverRev, firmRev)

        REM Close communication channel
        status = niSwitch_close(instr)
End Sub
```

Example 2-2

```
Private Sub vbMain()
         Dim instr As ViSession         'REM Communication Channel
         Dim status As ViStatus         'REM For checking errors


         Rem Begin by opening a communication channel to the instrument
         status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, instr)
         if (status < VI_SUCCEESS) Then
                Rem Error initializing interface ... exiting
                Exit Sub
         End If

         REM NOTE: For simplicity we will not show any other error checking

         REM Disconnect Channel 0 from the common (open the switch)

         status = niSwitch_Disconnect(instr, "com0", "ch0")

         REM Connect Channel 16 to the common (close the switch)
         status = niSwitch_Connect(instr, "com16", "ch16")

         REM Close communication channel
         status = niSwitch_close(instr)
End Sub
```

Example 2-3

```
Private Sub vbMain()
        Dim instr As ViSession          'Communication Channel
        Dim status As ViStatus          'For checking errors

        Rem Begin by opening a communication channel to the instrument
        status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, instr)
        if (status < VI_SUCCESS) Then
              Rem Error initializing interface ... exiting
              Exit Sub
        End If

        REM NOTE: For simplicity we will not show any other error checking

        REM Close switch #0
        status = niSwitch_Connect(instr, "com0", "ch0")
        status = niSwitch_WaitForDebounce(instr, 1000)

        REM INSERT CODE TO MAKE READING

        REM Open switch #0
        status = niSwitch_Disconnect(instr, "com0", "ch0")

        REM Close switch #1
        status = niSwitch_Connect(instr, "com0", "ch1")
        status = niSwitch_WaitForDebounce(instr, 1000)

        REM INSERT CODE TO MAKE READING

        REM Close communication channel
        status = niSwitch_close(instr)
End Sub
```

Example 2-4

```
Private Sub vbMain()
         Dim instr As ViSession          'Communication Channel
         Dim status As ViStatus          'For checking errors

         Rem Begin by opening a communication channel to the instrument
         status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, instr)
         if (status < VI_SUCCESS) Then
               Rem Error initializing interface ... exiting
               Exit Sub
         End If

         REM NOTE: For simplicity we will not show any other error checking

         REM Connect the Matrix Point (row=0, col=0)
         status = niSwitch_Connect(instr, "r0", "c0")

         REM Connect the Matrix Point (row=3, col=4)
         status = niSwitch_Connect(instr, "r3", "c4")

         REM Disconnect the Matrix Point (row=0, col=0)
         status = niSwitch_Disconnect(instr, "r0", "c0")

         REM Close communication channel
         status = niSwitch_close(instr)
End Sub
```

Example 2-5

```
Private Sub vbMain()
        Dim instr As ViSession          'Communication Channel
        Dim status As ViStatus          'For checking errors


        Rem Begin by opening a communication channel to the instrument
        status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE, instr)
        if (status < VISUCCESS) Then
               Rem Error initializing interface ... exiting
               Exit Sub
        End If

        REM NOTE: For simplicity we will not show any other error checking

        REM Turn off Continuous mode.  We want a one-shot scan
        status = niSwitch_SetContinuousScan(instr, VI_FALSE)


        REM CONFIGURE THE DMM TO TAKE 16 READINGS AND WAIT FOR A
        REM TRIGGER BEFORE STARTING EACH READING.  ALSO
        REM ASSERT A TRIGGER AFTER EACH READING


        REM Now scan...
        status = niSwitch_Scan(instr, "com0->ch0:15;")

        /* Wait for Scan to complete */
        status = niSwitch_WaitForScanComplete(instr, 5000)

        /* DOWNLOAD DATA FROM DMM */
        REM DOWNLOAD DATA FROM DMM

        REM Close communication channel
        status = niSwitch_close(instr)
End Sub
```

Example 5-1

```
Private Sub vbMain()
         Dim DMMinstr as ViSession      'Communication Channel
         Dim SWITCHinstr as ViSession   'Communication Channel

         REM DMM Variables
         Dim maximumTime as ViInt32     'Number of ms to wait for operation
         Dim sampleInterval as ViReal64 'Time waited after trigger
         Dim sampleCount as ViInt32     'Number of samples per trigger
         Dim trigCount as ViInt32       'Number of triggers to count
         Dim trigDelay as ViReal64      'Time to wait from trigger
         Dim arraySize as ViInt32       'Size of data array
         Dim arrayData as ViReal64 * 64 'Data from DMM
         Dim actualNumPoints as ViInt32 'Number of points returned from DMM

         Dim status as ViStatus

         REM Initialize DMM variables

         mximumTime = 5000
         sampleInterval = 0
         sampleCount = 1
         trigCount = 64
         trigDelay = 0.0
         ACminFreq = 20.0
         ACmaxFreq = 20000
         arraySize = 64

         Rem Begin by opening a communication channel to the instrument
         status = niSwitch_init("PXI0::16::INSTR", VI_TRUE, VI_TRUE,
                                 SWITCHinstr)
         if (status < VI_SUCCESS) Then
               Rem Error initializing interface ... exiting
               Exit Sub
         End If
         status = niDMM_init("DAQ::2::INSTR", VI_TRUE, VI_TRUE, DMMinstr)
         if (status < VI_SUCCESS) Then
               REM Error Initializing Interface...exiting
               niSwitch_close(SWITCHinstr)
               return -1;
```

```
REM NOTE: For simplicity we will not show any other error checking

REM Start by configuring the handshake lines
REM In this case, we are using TTL0 and 1 for triggers
REM for the PXI switches
status = niSwitch_ConfigureScanTrigger(SWITCHinstr, 0.0,
                                       NISWITCH_VAL_TTL0,
                                       NISWITCH_VAL_TTL1)

REM Provide the list of channels to scan and also say
REM what scan mode to use
status = niSwitch_ConfigureScanList(SWITCHinstr, "com0->ch0->15;",
                                  NISWITCH_VAL_BREAK_BEFORE_MAKE)

REM Set the Continuous mode
status = niSwitch_SetContinuousScan(SWITCHinstr, VI_TRUE)

REM CONFIGURE THE DMM TO TAKE 64 READINGS AND WAIT FOR A
REM TRIGGER ON TTL1 BEFORE STARTING EACH READING. ALSO
REM ASSERT A TRIGGER ON TLL0 AFTER EACH READING.

status = niDMM_SetPowerLineFrequency(DMMinstr, NIDMM_VAL_60_HERTZ)

status = niDMM_Configure(DMMinstr, NIDMM_VAL_DC_VOLTS,
                          NIDMM_VAL_AUTO_RANGE_ON,
                          NIDMM_VAL_5_5_DIGITS, ACminFreq,
                          ACmaxFreq)

status = niDMM_ConfigureMultiPoint(DMMinstr, trigCount,
                                    sampleCount,
                                    NIDMM_VAL_IMMEDIATE,
                                    sampleInterval)

status = niDMM_ConfigureTrigger(DMMinstr, NIDMM_VAL_TTL1,
                                 trigDelay)

status = niDMM_ConfigureMeasurementComplete(DMMinstr,
                                            NIDMM_VAL_TTL0,
                                            NIDMM_VAL_POS)

status = niDMM_Initiate(DMMinstr)
```

```
         REM Now scan 16 channels on the switch 4 times (16x4 = 64)...
         status = niSwitch_InitiateScan(SWITCHinstr)


         REM DMM DETECTION OF COMPLETE SCAN AND RETRIEVE DATA
         status = niDMM_FetchMultiPoint(DMMinstr, maximumTime, arraySize,
                                        arrayData, actualNumPoints)

         REM Stop the Switch from scanning
         status = niSwitch_AbortScan(SWITCHinstr)

         REM Close communication channel
         status = niSwitch_close(SWITCHinstr)
         status = niDMM_close(DMMinstr)

End Sub
```

B

Multiple Card Scanning

This appendix covers the unique features that affect scanning when you have multiple switch cards wired together to act as one large switch.

For example, if you wire together the analog bus connections of multiple NI 2501 switch cards, you can have *n* times 24 channels, where *n* is the number of switch cards wired together. However, from the NI-SWITCH driver's point of view, these are still unique switch cards with their own unique addresses. To handle scanning in these situations, you need to:

1. Count triggers in the various scan lists.
2. Set up timing information.
3. Configure triggers.

Scan Lists

To help you understand multiple scan lists, consider the case of multiple NI 2501 cards. The NI 2501 is a 24-channel FET multiplexer that supports the analog bus connections possible through the PXI terminal block or terminal-block wiring. In this example, assume we have three NI 2501 cards wired together to create a 72-channel FET multiplexer. If we wanted to do a simple scan of all three cards in order, the scan lists for each card would be as follows:

```
Card #1:  "ab0->ch0:23; <REPEAT 24>; <REPEAT 24>;"
Card #2:  "<REPEAT 24>; ab0->ch0:23; <REPEAT 24>;"
Card #3:  "<REPEAT 24>; <REPEAT 24>; ab0->ch0:23;"
```

Notice that the number 24 in the `REPEAT` command word is the exact number of channels scanned by the other cards. The above example used two `REPEAT` command words to make this clear. However, you could have combined the two command words.

Note ***This example shows direct connections from the input channels to the analog bus. Refer to the*** *Analog Bus* ***section in Chapter 4,*** *Manual Switch Control****, to see how this is done.***

To make sure the scan starts properly, call either the ***`Scan`*** or ***`Initiate Scan`*** operation, beginning with the switch cards that do not have any switch activity in the first element of the scan list—that is, card 3, then card 2, and then card 1. This way, the final call causes the switch to close, and subsequently causes the trigger that starts off the handshake.

Switch Timing

Another issue for you to consider is that all the switch cards working together as one switch need to have the same debounce times. This is already the case if you are using the same model, such as multiple NI 2501 switch cards. However, if you are using different models, you need to change the times to the *worst case* of all the switch cards.

The relevant attribute is:

- *`SCAN DELAY`*

Warning ***A system without uniform debounce times can result in dangerous race conditions that may not be controlled by break-before-make or break-after-make. These conditions can result in damage to the system or personal injury.***

Trigger Configuration

The final consideration when performing multiple card scanning is the trigger mapping. When a scan involves only a single switch card and a device, the handshake triggers are point-to-point and therefore very simple. However, when working with multiple switch cards, you must ensure that all cards can participate in the scan and can access the triggers. If the measurement and switch cards are all in the same system—such as a PXI digital multimeter (DMM) and PXI switch—the trigger mapping remains simple because the trigger lines of the chassis are all bused to each slot. In these cases, the trigger input and output can be the same for each switch card.

If you must use the front-panel triggers, the situation becomes more complicated, though still manageable. The NI-SWITCH driver handles this through some more advanced triggering attributes.

Let's begin with the ***`TRIGGER MODE`*** Attribute. This attribute tells the card whether it is operating in single, master, or slave mode. Refer to the online help for a full description of this attribute.

If the mode for this attribute is set to single (`NISWITCH_VAL_SINGLE`) you are using the switch card in a single-card scanning system and therefore can use the configurations as described in Chapter 5, *Scanning*. Master and slave modes are meant for multiple-card scanning.

The master switch card is the card where the external triggers are wired. If the triggers are coming from a backplane, the master is the *first* switch card in the system—typically in the leftmost slot, although you can choose other locations.

The master is responsible for propagating the triggers to the various slave cards. You use two more attributes to arrange this:

- ***MASTER SLAVE TRIGGER BUS***
- ***SLAVE MASTER SCAN ADVANCED BUS***

These attributes indicate which two backplane trigger lines to use to bus triggers between the master and slave cards. For example, consider a system with an external DMM and two PXI switch cards. The first switch card is the master and uses PXI TTL0 and TTL1 for trigger busing.

The following diagram illustrates this system.

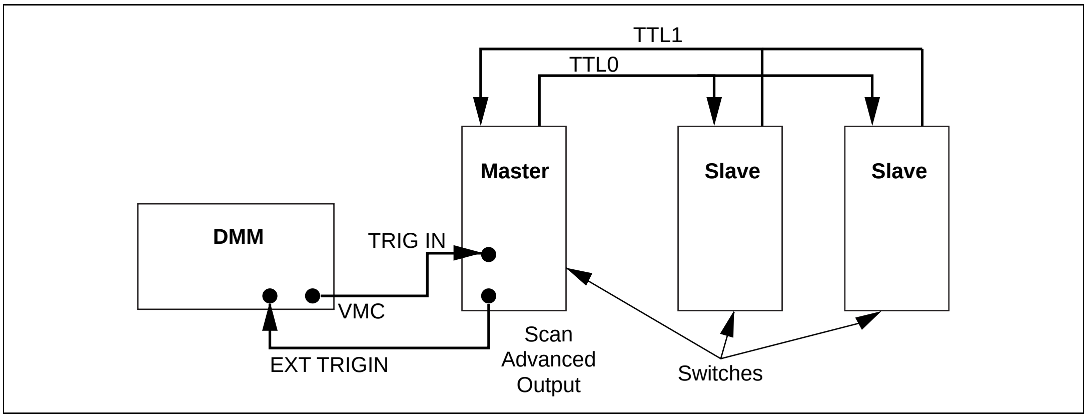


Figure B-1. External Trigger System

In this case, the attributes for both the master and slave cards would be set as follows.

Attribute	Value
TRIGGER INPUT	NISWITCH_VAL_EXTERNAL
SCAN ADVANCED OUTPUT	NISWITCH_VAL_EXTERNAL
MASTER SLAVE TRIGGER BUS	NISWITCH_VAL_TTL0
SLAVE MASTER SCAN ADVANCED BUS	NISWITCH_VAL_TTL1

Technically, you could set the ***Trigger Input*** and ***Scan Advanced Output*** attributes for the slave cards to anything since the driver ignores them.

To continue the example described earlier, let us examine the difference if the DMM is not external but rather is internal to the chassis and shares the same trigger lines. In this case, the DMM should use matching trigger lines.

The following diagram illustrates this example.

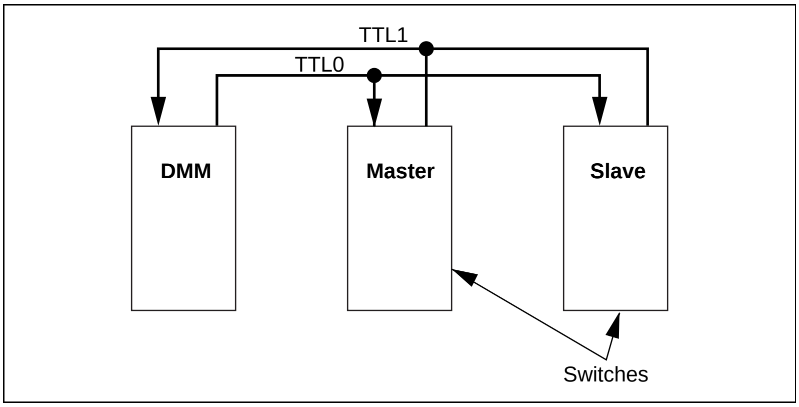


Figure B-2. Internal Trigger System

For this example, the DMM should be configured as follows:

Trigger	Value
DMM Volt Meter Complete Trigger	TTL0
DMM Trigger Input	TTL1

The master and slave cards would then be configured as follows.

Attribute	Value
TRIGGER INPUT	`NISWITCH_VAL_TTL0`
SCAN ADVANCED OUTPUT	`NISWITCH_VAL_TTL1`
MASTER SLAVE TRIGGER BUS	`NISWITCH_VAL_TTL0`
SLAVE MASTER SCAN ADVANCED BUS	`NISWITCH_VAL_TTL1`

C

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___) ________________Phone (___) ____________________________

Computer brand____________ Model ___________________Processor____________________

Operating system (include version number) ____________________________

Clock speed ______MHz RAM _____MB Display adapter ____________________

Mouse ___yes ___no Other adapters installed____________________________

Hard disk capacity _____MB Brand______________________________

Instruments used ____

____

National Instruments hardware product model_____________ Revision ____________________

Configuration ____

National Instruments software product___________________ Version ____________________

Configuration ____

The problem is: ____

____

____

____

____

List any error messages: ____

____

____

The following steps reproduce the problem: ____________________________

____

____

____

____

NI-SWITCH Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision ____________________

Interrupt level of hardware ____________________

Base address of hardware ____________________

Programming choice ____________________

National Instruments software and revision number____________________

____

Other boards in system ____________________

Base address of other boards ____________________

Interrupt level of other boards ____________________

Other Products

Computer make and model ____________________

Microprocessor ____________________

Clock frequency or speed ____________________

Type of video board installed ____________________

Operating system version ____________________

Operating system mode ____________________

Programming language ____________________

Programming language version ____________________

Other boards in system ____________________

Base address of other boards ____________________

Interrupt level of other boards ____________________

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *NI-SWITCH™ Software User Manual*

Edition Date: June 1998

Part Number: 321900A-01

Please comment on the completeness, clarity, and organization of the manual.

____

____

____

____

____

____

____

If you find errors in the manual, please record the page numbers and describe the errors.

____

____

____

____

____

____

____

Thank you for your help.

Name ____

Title ____

Company ____

Address ____

____

E-Mail Address ____

Phone (___) ______________________________ Fax (___) ______________________________

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meanings	Value
p-	pico	10^{-12}
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}
G-	giga-	10^{9}
t-	tera-	10^{12}

A

address string	a string (or other language construct) that uniquely locates and identifies a resource. VISA defines an ASCII-based grammar that associates strings with particular physical devices and VISA resources.
API	Application Programming Interface. The direct interface that an end user sees when creating an application. In VISA, the API consists of the sum of all of the operations, attributes, and events of each of the VISA resource classes.
attribute	a value within an object or resource that reflects a characteristic of its operational state.

B

breakpoint	a specified point in program code where the program pauses to perform some action; a breakpoint interrupt can be added to a scan list for debugging or other special needs.

bus	the group of conductors that interconnect individual circuitry in a computer. Typically, a bus is the expansion vehicle to which I/O or other devices are connected. Examples of PC buses are the ISA and PCI bus.

C

C	Celsius
channel	pin or wire lead to which you apply or from which you read the analog or digital signal
common	a channel that is typically the *output* of a switch module
communication channel	the same as *session*. A communication path between a software element and a resource.
contact bounce	the intermittent switching that occurs when the movable metal parts of a relay make or break contact
controller	an entity that can control another device(s) or is in the process of performing an operation on another device

D

debounced	indicates when the contact bounce has ended. *See* contact bounce.
device	an entity that receives commands from a controller. A device can be an instrument, a computer (acting in a non-controller role), or a peripheral (such as a plotter or printer).
digital multimeter	a multifunction meter used to make measurements such as voltage, current, resistance frequency, temperature, and so on
DLL	Dynamic Link Library. Same as a *shared library* or *shared object*. A file containing a collection of functions that can be used by multiple applications. This term is usually used for libraries on Windows platforms.
DMM	*See* digital multimeter.
drivers/driver software	software that controls a specific hardware device such as a switch card

E

external trigger	a voltage pulse from an external source that triggers an event such as A/D conversion. *External* typically means external from the chassis.

F

FET	Field Effect Transistor

H

handshaking	the use of two trigger lines between two instruments, such as a switch and a DMM, to synchronize their actions
Hz	hertz—the number of scans read or updates written per second

I

I/O	input/output—the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces
instrument	a device that accepts some form of stimulus to perform a designated task, test, or measurement function. Two common forms of stimuli are message passing and register reads and writes. Other forms include triggering or varying forms of asynchronous control.
instrument driver	a set of routines designed to control a specific instrument or family of instruments, and any necessary related files for LabWindows/CVI or LabVIEW
interface	a generic term that applies to the connection between devices and controllers. It includes the communication media and the device/controller hardware necessary for cross-communication.
interrupt	a condition that requires attention out of the normal flow of control of a program
Intelligent Virtual Instrument	an advanced architecture for instrument drivers that includes features such as simulation and state caching

ISA	Industry Standard Architecture
IVI	*See* Intelligent Virtual Instrument.

L

latching relay	a relay that maintains its state when power is removed
lock	a state that prohibits sessions other than the session(s) owning the lock from accessing a resource

M

master switch card	the first switch card in a multi-card scan. It is responsible for passing the triggers from the instrument to and from the slave switch card.
matrix	superset of multiplexer; consists of connected rows and columns that allows for a direct connection from any row to any column
multiplexer	a switch module designed to take multiple *input* channels and allow the user to select one channel as the *output*

N

NI-SWITCH	an IVI-based instrument driver that supports the National Instruments line of switch cards
non-latching relay	a relay that requires constant power to maintain its state

O

operation	an action defined by a resource that can be performed on a resource. In general, this term is synonymous with the connotation of the word *method* in object-oriented architectures. Also known as a *function* in C or a *VI* in LabVIEW.

P

process	an operating system element that shares a system’s resources. A multi-process system is a computer system that allows multiple programs to execute simultaneously, each in a separate process environment. A single-process system is a computer system that allows only a single program to execute at a given point in time.
property node	a primitive in LabVIEW you can use to read or write attributes
PXI	PCI with extensions for instrumentation

R

random scanning	scanning the channels in a multiplexer in any order
relay	a switch that connects or disconnects the signal to a common through the physical movement of a metal arm
resource name	*See* address string.
row and column	another word for channel, used to describe the names of channels for a matrix

S

s	seconds
scan	a sequence of channel connections typically controlled by triggers
scan advanced trigger	the trigger generated by the switch card when scanning. The trigger occurs after the switch card has closed a switch and the switch has settled.
scan list	a list of channels supplied to NI-SWITCH that indicates the order in which channels will be scanned
scanner	*See* multiplexer.
SCXI	Signal Conditioning eXtensions for Instrumentation—the National Instruments product line for conditioning low-level signals within an external chassis near sensors so only high-level signals are sent to DAQ boards in the noisy PC environment

session	the same as *communication channel*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
settling time	the amount of time required for a voltage to reach its final value within specified limits. *See* debounced.
simulation mode	a feature of the IVI architecture. A user can open a session to a simulated switch module and develop code without having the switch module physically present.
slave switch card	the switch cards other than the master switch card in a multi-card scan
soft front panel	a graphical program included with NI-SWITCH that you can use to interactively control the switch
state caching	a feature of the IVI architecture. The driver can maintain the state of the switch module in software to reduce unnecessary communication with the switch module.

T

TBX	Terminal Block Extension
terminal block	an accessory containing wire connection points, typically screw terminals.
tree	*See* multiplexer.
trigger	any event that causes or starts some form of data capture

V

virtual instrument	(1) a combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument; (2) a LabVIEW software module (VI), which consists of a front-panel user interface and a block diagram program
VISA	Virtual Instrument Software Architecture. This is the general name given to this product and its associated architecture. The architecture consists of two main VISA components: the VISA resource manager and the VISA resources.
VXI	VMEbus Extensions for Instrumentation

W

wire	data path between nodes in LabVIEW

Index