Computer-Based Instruments

NI-SWITCH
Software User Manual

February 2001 Edition
Part Number 370154B-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 514 694 8521,
China (Shanghai) 021 6555 7838, China (ShenZhen) 0755 3904939, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico 5 280 7625, Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

CVI™, LabVIEW™, National Instruments™, ni.com™, PXI™, and SCXI™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

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

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

Conventions

The following conventions are used in this manual:

♦ The ♦ symbol indicates that the following text applies only to a specific product, a specific operating system, or a specific software version.

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

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

This icon denotes a warning, which advises you of precautions to take to avoid being electrically shocked.

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

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

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions. In addition, this manual uses this font as a naming convention to jointly refer to LabVIEW VIs and C-language function calls. For example, `Connect`, when shown with this font, refers both to the LabVIEW VI niSwitch Connect Channels and to the C function `niSwitch_Connect`. Refer to Appendix C, *Common Names Table*, for more information.

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

Contents

Chapter 1
Introduction

Chapter 2
API Overview

Chapter 3
Introductory Programming Examples

Chapter 4
Manually Controlling Switches

Chapter 5
Scanning

1 Introduction

This manual describes how you can use the NI-SWITCH driver to program your National Instruments switch device.

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 *Getting Started* section.

Use your switch device user manual to learn about the electrical and mechanical aspects and features of your National Instruments switch device.

Note Read the `Readme.htm` file that was installed with the instrument driver for last-minute changes and updates.

In addition to this manual and your switch device user manual, you can find useful information in the online help files. These electronic documents (`.hlp` files) give detailed information on the operations and attributes of NI-SWITCH. Refer to the Where to Start document that came with your hardware for instructions about setting up your system.

For the latest versions of drivers, manuals, and example programs, visit `ni.com/instruments` for free downloads.

Note This manual uses a naming convention to jointly refer to LabVIEW VIs and C-language function calls. For example, `Connect`, when shown with this font, refers both to the LabVIEW VI niSwitch Connect Channels and to the C function `niSwitch_Connect`. Refer to Appendix C, *Common Names Table*, for more information.

Background

NI-SWITCH is designed to be an easy-to-use software interface for National Instrument switch products. This driver is based on two industry standards for instrument drivers—VXI*plug&play* and Interchangeable 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 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.

IVI

In 1997, National Instruments promoted the creation of the IVI Foundation, an open consortium of companies chartered with the purpose of defining software standards for Interchangeable Virtual Instruments (IVI). As a result, the IVI Foundation (`www.ivifoundation.org`) has introduced an instrument driver model that is not only VXI*plug&play* compliant but also extends the functionality to include many new features, such as state-caching and simulation modes, which users had requested. 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.

Getting Started

You need the following items:

- ❑ Appropriate National Instruments switching hardware, such as the NI 25*XX* series for PXI, SCXI-1127/1128, SCXI-1129, or SCXI-1160/1161/1163R. Refer to the `Readme.htm` file for a complete list of devices you can use with the version of the driver you have.
- ❑ NI-SWITCH instrument driver software

Installing the Software

You can control your SCXI switch module programmatically in an application development environment (ADE) using NI-SWITCH. The supported ADEs include LabVIEW, Measurement Studio, Visual Basic, and C or C++ environments. To install NI-SWITCH, complete the following:

1. Install your ADE if you have not done so already.
2. Insert your NI-SWITCH software CD into your CD-ROM drive. The installation window appears automatically.

Note If the installation window does not appear, double-click the **My Computer** icon on your PC desktop. Find the CD drive, and double-click the `install.exe`.

The NI-SWITCH and NI-DMM installation window offers three choices:

- **Install NI-DMM**—Choose this selection if you want to install only the software for the DMM.
- **Install NI-SWITCH**—Choose this selection if you want to install only the software for the switches.
- **Install NI-SWITCH and NI-DMM**—Choose this selection if you want to install the software for both the DMM and the switches.

3. Click the **Install NI-SWITCH** option.
4. To install the instrument driver, Soft Front Panel(s), and ADE examples, choose **Programmatic and Interactive Support**. To install only the Soft Front Panel(s), choose **Interactive Support Only**.

Note The **Interactive Support Only** choice does *not* allow you to program the instrument with any programming languages.

5. For advanced users only—when installing NI-SWITCH, notice the **Development Environments** panel.
   - If you click the **Advanced** button on this panel, you can custom install National Instruments drivers such as NI-DAQ, NI-IVI, and NI-VISA.
   - If you do *not* want to install certain drivers, click **Advanced** and uncheck the driver(s) you do not want installed.

If a driver is already unchecked, your computer has the same driver or a newer version of the driver already installed.

Note If a newer version of a driver is present on your PC, the installer does *not* overwrite the driver.

After completing the software installation, turn off your computer. Refer to your hardware user manual for instructions on installing your hardware.

2

API Overview

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.

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 VXI*plug&play*, VISA, and IVI technology. Due to advances in these technologies, several object-oriented concepts exist in the API. When you consider a description of an object, such as a car, computer, or switch device, it has three groups of information associated with it, as described below:

- The first group is *attributes*. Attributes give state information about the object. For example, attributes of a car include its color, the number of doors, and whether or not it is currently running. For a switch device, attributes include the number of channels, the bandwidth of the device, and whether or not the switches on the device are debounced. In addition, attributes can also control features of the switch such as controlling which trigger lines the switch device should use.
- The second group is *operations*. Operations 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 device, operations include opening or closing a switch.
- The third group is *sessions*, or *handles*. Sessions 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 device 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 device being connected to a measurement device and the switch device passing signals through to the measurement device. This reference is not meant to restrict the use of the switch device, 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 device.

Switch Overview

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 your point of view, the switch is no more than a way to connect signal paths. If you 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*. The NI-SWITCH driver was designed from this point of view.

Simple Multiplexer

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 program the driver to connect the two channels as shown in Figure 2-1.

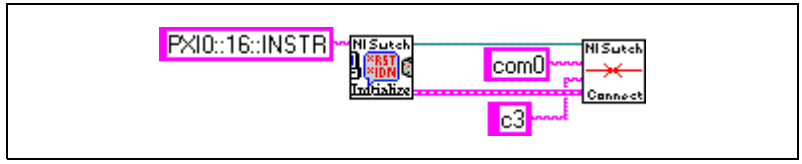

Figure 2-1. Connect Channel to Common Block Diagram

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

Matrix

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 between Figure 2-1 and Figure 2-2, which illustrates how to connect row 5 to column 3.

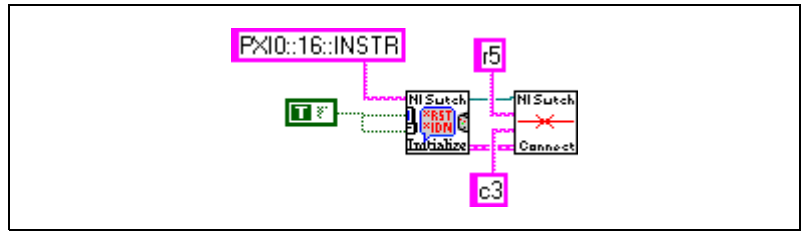

Figure 2-2. Connect Row with Column Block Diagram

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

General-Purpose Switch Module

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 × 1 multiplexer, and a 1-Form C switch as a 2 × 1 multiplexer. In the case of a 1-Form C switch, its channels are often called COM (common), NC (normally closed), and NO (normally open). The connect operation is shown in Figure 2-3.

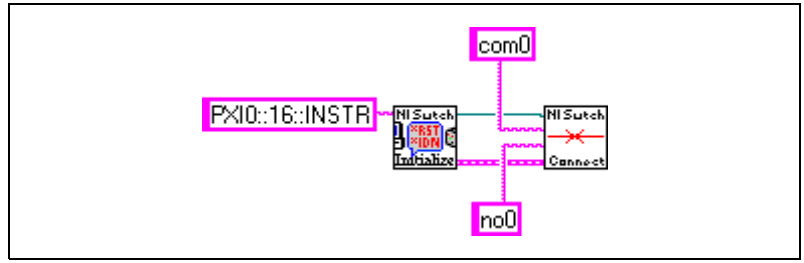

Figure 2-3. Connecting a General-Purpose Switch Module Block Diagram

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

Using a Session for Communication

NI-SWITCH is a *scalable* driver, which means it can communicate with 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 `Initialize`, 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. The session is the central component to communicating with the driver and hardware.

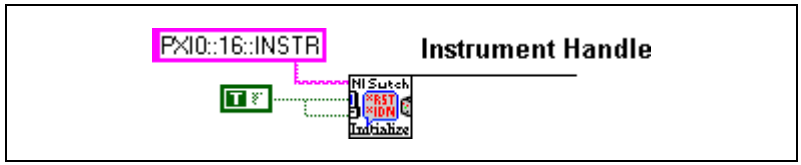

Figure 2-4. Initialize Block Diagram

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

This code opens communication with a PXI switch device at address 16. The session is returned in the last parameter, in this case in the variable *instr*.

As an alternative to `Initialize`, 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 NI-SWITCH in simulation mode, in which you can run programs without the switch hardware being connected to the computer. To open a session to the simulation driver, you would use `Initialize With Options` as shown in Figure 2-5.

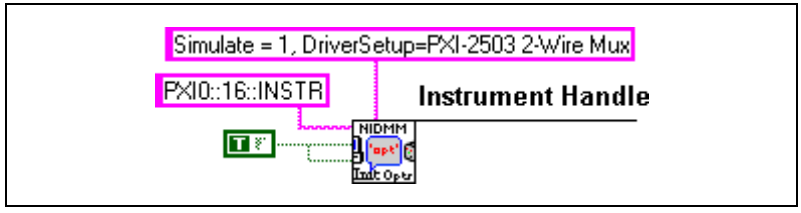

Figure 2-5. Initialize With Options Block Diagram

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

To communicate with the hardware, use this session as the first parameter of every operation from then on, as shown in Figure 2-6.

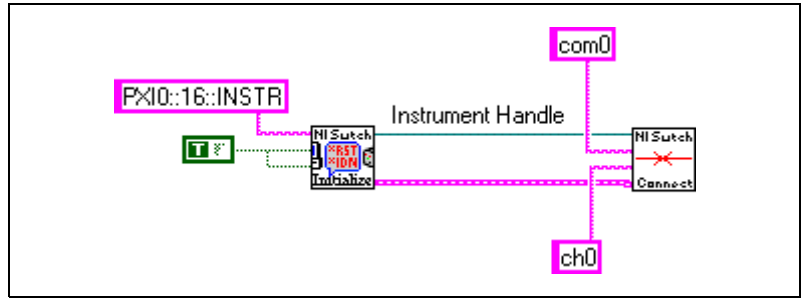

Figure 2-6. Use the Instrument Handle in Your Application Block Diagram

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

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

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 for use in Visual Basic and C/C++. When locking is required in LabVIEW, use LabVIEW-specific methods.

Using Attributes

You can use attributes to get information about the state of the device, or to set the state of the device. For example, by retrieving an attribute from the driver, you can determine whether the switches on the device have debounced, or you can set the source of a trigger. Some NI-SWITCH *helper* operations even let you use attributes without accessing them directly. For example, `Is Debounced` also tells you whether or not the switches on the device have settled, and `Configure Scan Trigger` sets up the necessary trigger parameters in a single call.

Accessing Attributes in LabVIEW, C, and Visual Basic

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_`.

Figures 2-7 and 2-8 show how to get or set attributes in LabVIEW and C:

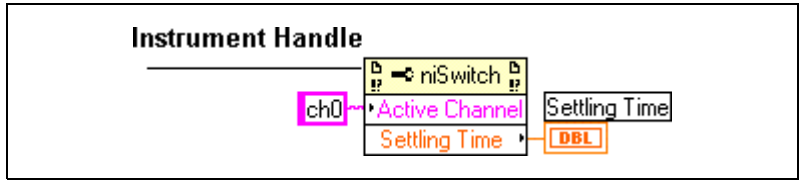

Figure 2-7. Get Active Channel Attribute Value Block Diagram

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

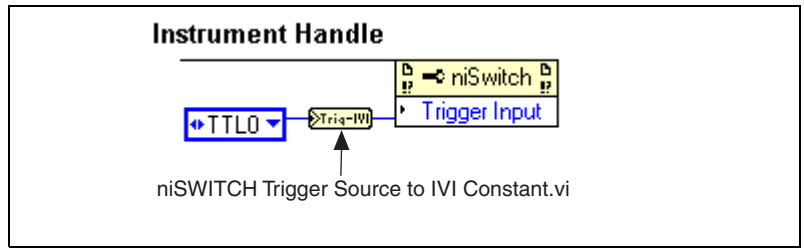

Figure 2-8. Set Trigger Input Attribute Block Diagram

```
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, each data type has a unique function (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 device, or local to each channel of the device. 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.

Using the C Attributes for Querying and Controlling

The *NI-SWITCH C Reference 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.

Using Read-Only State Attributes to Query the Switch Device

The first group of attributes is the read-only state attributes. You can use these attributes to query the switch device for basic information about the switch. This makes it possible for a program to be scalable across different devices because the program can 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 operations. Therefore, getting the bandwidth value for the first switch in a multiplexer does not tell you the bandwidth through the device, but only through that specific switch.

Using Operation Attributes to Control Basic Operation of the Switch

Another group of attributes controls basic operation of the switch device. 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*.

Using Operations in LabVIEW, C, and Visual Basic

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 NI-SWITCH, including access to attributes (through the Get and Set operations) for C and Visual Basic users. Remember, in LabVIEW, you use property nodes to access attributes.

Consult the *NI-SWITCH C Reverence Help* or the *NI-SWITCH VI Reference Help* for more information on NI-SWITCH operations.

Overview of VXI*plug&play* Required Operations

Some operations are required for VXI*plug&play*.

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 `Initialize` can perform a complete reset on the device.

Reset

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

Remember that the definition of this state can vary between latching and non-latching relay devices. For the non-latching devices, 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 device. However, all latching devices, when reset, return to their reset position.

Self Test

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

Error Query and Error Message

All operations return status information that indicates whether or not the operation executed successfully. However, you can use `Error Query` to retrieve the status code returned by the last operation called. `Error Message` translates an NI-SWITCH status code into a text message, which can be displayed via a pop-up window.

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.

3

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.

These examples 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 2000/NT/Me/9*x* programming environments. You can find equivalent examples for this environment in Appendix A, *Microsoft Visual Basic Examples*.

The program shown in Example 3-1 is very straightforward. Later examples in this chapter repeat its basic structure as additional features are introduced.

- C examples—Any code that is different from Example 3-1 in the subsequent examples appears in **`bold text`**.

Basic Startup

NI-SWITCH uses standard initialization and close routines at the beginning and end of the program. Example 3-1 shows how to get a handle to the instrument and retrieve information about the state of the hardware.

Example 3-1. Initialization

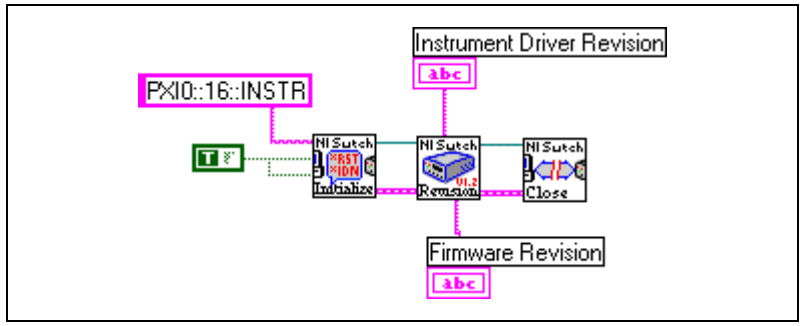

Figure 3-1. Initialize Example Block Diagram

```
#include "niswitch.h"

int main (void)

{
    ViSession SWITCHinstr;      /* 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,&SWITCHinstr);
   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(SWITCHinstr, driverRev, firmRev);

   /* Close communication channel */
   status = niSwitch_close(SWITCHinstr);
   return 0;

}
```

Discussion

Example 3-1 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—*SWITCHinstr*—is a session, or communication channel. This variable 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 device 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 device).

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 `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 device but also handle situations where you need multiple switches to connect different channels—such as connecting two columns in a matrix.

Example 3-2 shows how to connect channels on a switch device. For this example, assume you have a general-purpose relay device 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: an input (*ch* for channel) and an output (*com* for the common).

Figure 3-2 illustrates Form A, B, and C switches.

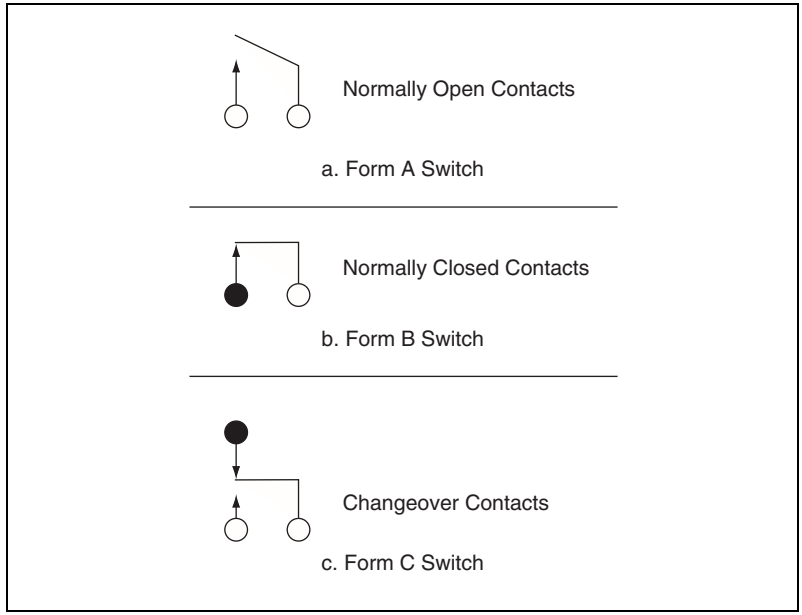

Figure 3-2. Form A, B, and C Switches

Example 3-2. General-Purpose Switches

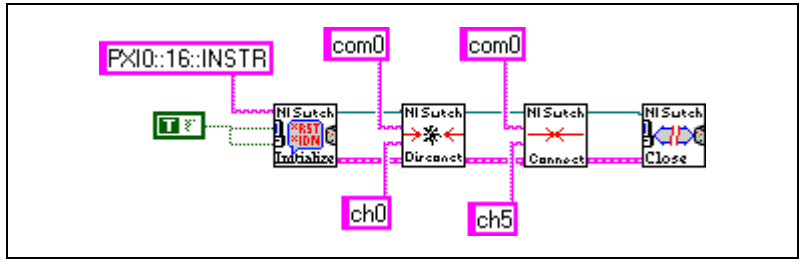

Figure 3-3. General-Purpose Switches Block Diagram

Note C code Examples 3-2 through 3-5 use **bold text** to distinguish lines of code that are different from Example 3-1.

```
#include "niswitch.h"

int main (void)
{
    ViSession SWITCHinstr;     /* 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,&SWITCHinstr);
   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(SWITCHinstr, "com0", "ch0");

    /* Connect Channel 16 to the common (close the switch) */
   status = niSwitch_Connect(SWITCHinstr, "com16", "ch16");

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

Discussion

Example 3-2 breaks down into the following parts:

- The program begins and ends with the same steps as described in Example 3-1. Refer to that example for more information on parts not in bold.
- 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 operation merely opens switch 0. There is no significance to which switch is connected to channel 1 or channel 2 input as they are interchangeable.
- The second command is `Connect`, which connects the channels of com16 and ch16. Again, this operation is basically closing switch 16 for the general-purpose switch.

Manual Scanning

Example 3-3 shows how to use the instrument driver to control a multiplexer. In Figure 3-4, notice the part of the VI that is marked “Take measurement device, and you need to program it.

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 short-circuited together.

Example 3-3. Multiplexer

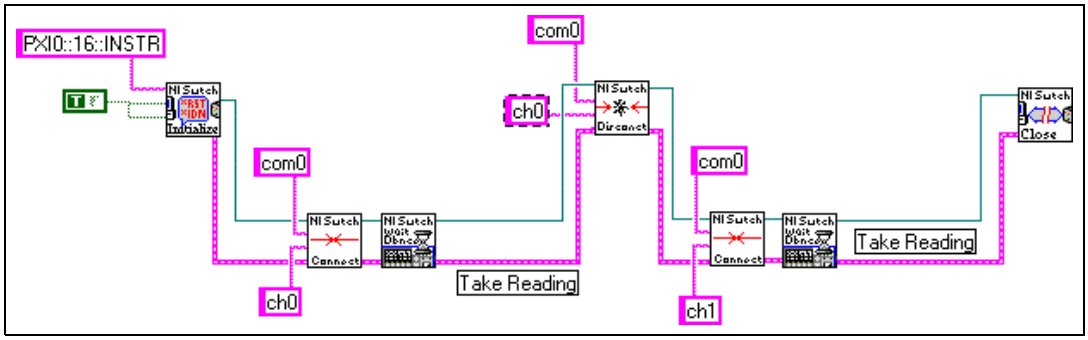

Figure 3-4. Multiplexer Control Block Diagram

```
#include "niswitch.h"

int main (void)
{
ViSession SWITCHinstr;/* 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,&SWITCHinstr);
   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(SWITCHinstr, "com0", "ch0");
   status = niSwitch_WaitForDebounce(SWITCHinstr, 1000);

    /* INSERT CODE TO MAKE READING */
```

```
    /* Open switch #0 */
    status = niSwitch_Disconnect(SWITCHinstr, "com0", "ch0");
    /* Close switch #1 */
    status = niSwitch_Connect(SWITCHinstr, "com0", "ch1");
    status = niSwitch_WaitForDebounce(SWITCHinstr, 1000);
    /* INSERT CODE TO MAKE READING */
    /* Close communication channel */
    status = niSwitch_close(SWITCHinstr);
    return 0;
}
```

Discussion

Example 3-3 breaks down into the following parts:

- The program begins and ends with the same steps as described in Example 3-1. Refer to that example for more information on steps not in bold.
- The first command is `Connect`. As described in Example 3-2, this operation connects the first channel to the output (com) 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 (1 s) as its **timeout** parameter. Notice that this is the default value for LabVIEW, so it is not wired.
- Next, the measurement device takes a reading from the output of the multiplexer.
- 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 `Wait For Debounce`. Opening channel 0 before closing channel 1 ensures that the two channels do not short-circuit together, as long as they are on the same device. However, since your concern is only that the device settles after channel 0 opens and channel 1 closes, you do not need to spend 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 devices 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 short-circuits.

- 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*. NI-SWITCH uses special channel strings for the matrix, but the general operation is the same because NI-SWITCH focuses on creating connections rather than opening and closing switches.

Example 3-4. Matrix

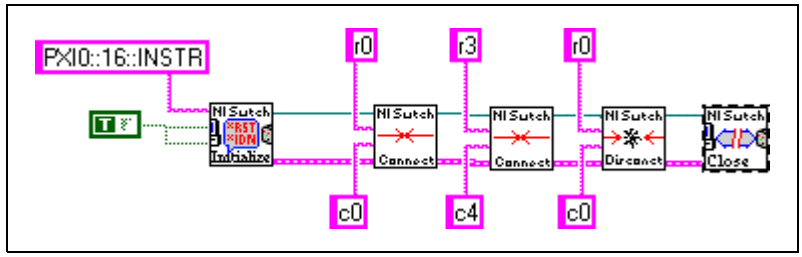

Figure 3-5. Matrix Mode Control Block Diagram

```
#include "niswitch.h"

int main (void)
{
    ViSession SWITCHinstr;      /* 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,&SWITCHinstr);
   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(SWITCHinstr, "r0", "c0");

    /* Connect the Matrix Point (row=3, col=4) */
```

```
    status = niSwitch_Connect(SWITCHinstr, "r3", "c4");
    /* Disconnect the Matrix Point (row=0, col=0) */
    status = niSwitch_Disconnect(SWITCHinstr, "r0", "c0");
    /* Close communication channel */
    status = niSwitch_close(SWITCHinstr);
    return 0;
}
```

Discussion

Example 3-4 breaks down into the following steps:

1. The program begins and ends with the same steps as described in Example 3-1. 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. In this situation, you can use scanning. Using scanning, you download the set of switches to open and close into the memory of the device. The scanning process then uses triggers to communicate between the measurement (or source) device and the switch device itself.

Example 3-5 shows how to scan 16 channels with a scanning digital multimeter (DMM). For more details of the scan list syntax itself, refer to the *Preparing a Scan List String* section in Chapter 5, *Scanning*.

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

Example 3-5. Scanning

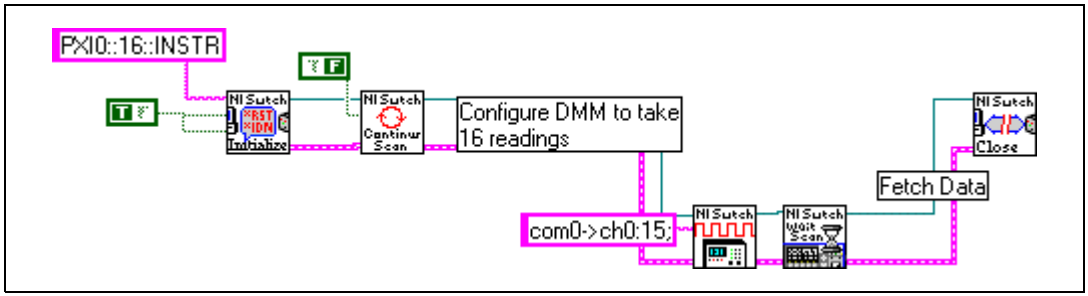

Figure 3-6. Scanning Switches with a DMM Block Diagram

```
#include "niswitch.h"

int main (void)
{
    ViSession SWITCHinstr;      /* 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,&SWITCHinstr);
   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(SWITCHinstr, 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;",NISWITCH_VAL_SWITCH_INITIATED);

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

    /* DOWNLOAD DATA FROM DMM */

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

Discussion

Example 3-5 breaks down into the following steps:

1. The program begins with the same steps as described in Example 3-1. Refer to that example for more information on steps not in bold.
2. At this point, you set the continuous mode to **False**. This mode indicates whether the switch device 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, you are using a scanning DMM. The code necessary is not included because that is dependent on the DMM. However, the set of steps necessary to configure the DMM are as follows:
   a. First, you configure the DMM to take readings only when it detects a trigger on its trigger input.
   b. Next, you configure it to generate triggers after it takes a reading.
   c. Finally, you tell the DMM to take 16 measurements.
4. You can now initiate the scan. NI-SWITCH has several operations for performing a scan, but the simplest one is `Scan`. Here you prepare a list of which channels to scan and in what order. The operation automatically programs the switch device and closes the first entry in the scan list. Because the switch device is configured to generate a trigger when a switch is closed, the first switch starts the handshaking. Refer to Chapter 5, *Scanning*, for more information about scanning operations and the scan list syntax.
5. `Scan` automatically returns the control to the program 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.

Note niSwitch Wait for Scan to Complete is only valid for PXI and cannot be used with SCXI.

6. The last step is to retrieve the measurements from the DMM when the scanning is complete.

4

Manually Controlling Switches

This chapter describes operations you can use to control a switch device 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.

Connect and Disconnect

Connect and disconnect operations perform standard connections between different switch channels. These operations make creating an application with the switch driver quick and simple.

General-Purpose Switch Topologies

The general-purpose topology refers to a switch device containing multiple switches that are completely independent of one another. Examples of general-purpose switch devices are the SCXI-1160 (16-channel switch) and the SCXI-1161 (8-channel, high-power switch). These National Instruments devices 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 devices to use. Below are LabVIEW and C examples of opening and closing switches on a general-purpose switch device.

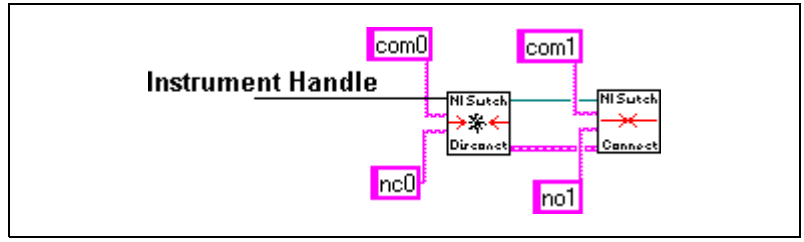

Figure 4-1. Open/Close General-Purpose Switch Block Diagram

```
/* 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 `Initialize`, 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 devices number their channels starting from zero. Refer to the hardware information contained in the switch device's user manual for a list of the channel numbers for the specific switch device.

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 are 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 device. For example, the NI 2503 is a 24-channel, 2-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, 2-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:

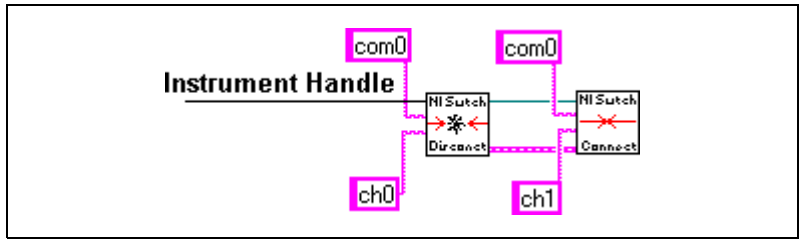

Figure 4-2. Connect/Disconnect on a Multiplexer/Scanner Switch Block Diagram

```
/* 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 `Initialize`, which the program must call before issuing any other operation to the hardware. The channel name parameters indicate which channel(s) to activate.

Matrixes

The final topology covered in this manual is the *matrix*. The matrix allows you to connect any input (row) to any output (column). 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 4 × 8 matrix could have a DMM, oscilloscope, function generator, and power supply connected to the four columns and could have eight 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 matrixes. Below are two examples of the operations taken from Chapter 3, *Introductory Programming Examples*.

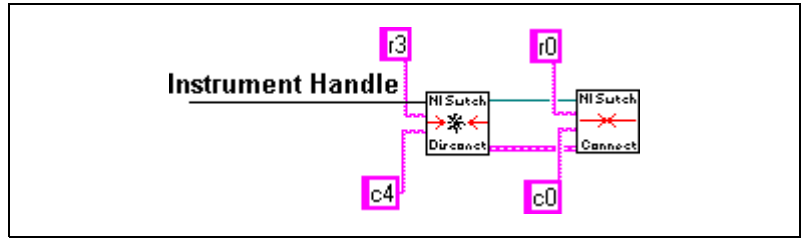

Figure 4-3. Connect/Disconnect Channels in a Matrix Topology Block Diagram

```
/* 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 `Initialize`, which must be called before any other operation to the hardware. The row and column names indicate which connections to make.

Reset

Chapter 2, *API Overview*, discussed two ways to reset the switch device to its powered-on state. These two ways 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 `Disconnect All` 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 short-circuiting 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 shown in Figure 4-4.

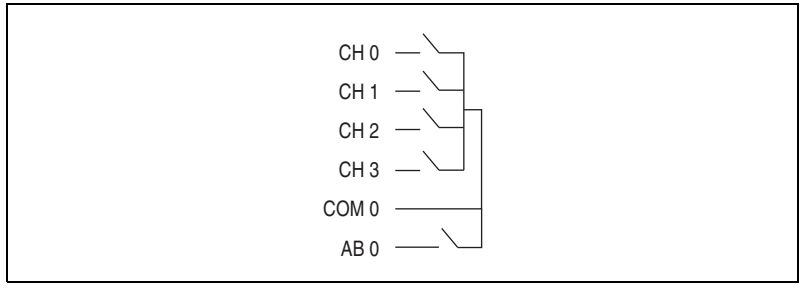

Figure 4-4. Analog Bus Connected to Common

For this reason, connecting to the analog bus can be a two-step process, as shown in Figure 4-5.

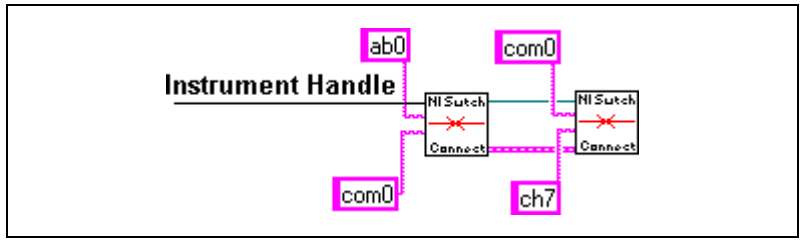

Figure 4-5. Connect Channel 7 to Analog Bus Block Diagram

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. `Connect` can close multiple switches so that it can make the path, but it cannot do this if it means going through extra channel switches.

5

Scanning

This chapter discusses the building blocks of scanning, such as scan lists and trigger configuration. In addition, this chapter includes scanning examples with either software triggers or with hardware timed scanning.

Overview

In many cases, you can control the switches on a switch device by supplying a list of channels to scan. Multiplexers, general-purpose, and matrix devices can use scanning to sequence through a set of patterns. You can program these patterns of connections using a scan list. The switch driver converts the scan list to the appropriate hardware commands and downloads them into memory located on the switch module. Once scanning has been initiated, the switch module executes the scan list independent of the host CPU.

The measurement device and the switch module use handshaking to control timing. Simply stated, when the measurement device finishes a measurement, it sends a digital trigger to the switch module. The switch module then opens and/or closes switches as indicated by the scan list. After these switches settle, the switch module sends a digital trigger to the measurement device, indicating that the device can make another measurement.

Other triggering methods, such as synchronous scanning and software triggering, are also possible. These methods are described in the *Configuring the Triggering Options* section. Refer to your measurement device and switch module hardware documentation for additional triggering information.

Notes All PXI modules can perform scanning, and the SCXI modules have varying levels of support for scanning. Refer to Chapter 6, *Using NI-SWITCH with SCXI*, for more information.

Whenever the switch device is scanning (`Is Scanning` 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 return an error if executed during scanning.

Preparing a Scan List String

The first step in scanning is to tell the driver what connections to make and in what order. You do this by preparing a *scan list string*, which is then parsed by the driver. Pass this string to the driveby setting the scan list attribute or by calling `scan`.

Using Basic Scan List Syntax

This section gives some basic syntax and examples of scan strings and describes how the scan list entries are interpreted when Scan Mode is set to No Action. This interpretation changes slightly when the Scan Mode is set to Break Before Make. This change, along with other time-saving and advanced syntax features, is described in the *Using Advanced Scan List Syntax* section.

You can use the basic scan list syntax described in this section to define almost any possible scan list by combining scan list characters to form scan list entries, then combining entries to form scan lists.

Connect Action

Connect actions instruct the driver to connect channels together. The syntax is `channelName->channelName`. White space is allowed on either side of the `->` characters.

Table 5-1. Connect Action Examples

String	Meaning
`a->b`	Connect a to b
`a -> b`	Same as above

Disconnect Action

Disconnect actions instruct the driver to disconnect channels. The syntax is a tilde (~) character followed by a connect action. White space is allowed after the tilde.

Table 5-2. Disconnect Action Examples

String	Meaning
`~a->b`	Disconnect a from b
`~ a->b`	Same as above

Switch Action

A switch action is either a connect action or a disconnect action and is defined only for convenience in describing the scan list syntax. The connect action syntax is *chanName->chanName*. The disconnect action syntax is *~chanName->chanName*.

Connection Separator

The connection separator is the ampersand (`&`) character. This character separates two switch actions. The connection separator does not ensure that the switch actions it separates execute in any particular order.

Table 5-3. Connection Separator Example

String	Meaning
`a->b & c->d`	Connect a to b and c to d

Sequence Separator

The sequence separator is a double ampersand (`&&`). Similar to a connection separator, a sequence separator separates two switch actions. A sequence separator can also have a switch action on its left and the end of the scan list on its right (useful when Scan Mode is set to No Action and Continuous Scan is set to True). The sequence separator instructs the driver that all currently switching switches must settle before performing the switch action on the right.

Table 5-4. Sequence Separator Example

String	Meaning
`a->b && c->d`	Connect a to b, wait for debounce, then connect c to d

Action Separator

An action separator is either a connection separator (`&`) or a sequence separator (`&&`). Action separators are defined only for convenience in describing the scan list syntax.

Scan List Entry

A scan list entry is made up of zero or more switch actions separated by action separators. A scan list entry is terminated by a semicolon or by the end of the scan list. Refer to Table 5-5 for scan list entry examples.

If the scan list entry contains at least one connect action, then a scanner advanced signal (shown as `<sa>` in Table 5-5) is generated after the switch actions specified have debounced. If the entry is terminated with a semicolon, then after performing the switch actions and generating the scanner advanced signal (if needed), the switch module waits for a trigger (shown as `<wft>` in Table 5-5).

Table 5-5. Scan List Entry Examples

String	Meaning
`a->b;`	Connect a to b, wait for debounce, then send <sa>, then <wft>
`a->b`	Connect a to b, wait for debounce, then send <sa>
`a->b & c->d;`	Connect a to b and c to d, wait for debounce, then send <sa>, then <wft>
`a->b && c->d;`	Connect a to b, wait for debounce, then connect c to d, wait for debounce, then send <sa>, and finally <wft>
`a->b & ~c->d;`	Connect a to b and disconnect c from d, wait for connect and disconnect to debounce, then send <sa>, then <wft>
`~c->d;`	Disconnect c from d, then <wft>
`;`	<wft>

The last two examples in Table 5-5 do not generate an `<sa>` since no connection actions are in these entries.

A scan list entry containing a <wft> without a <sa> can be useful when doing multimodule scanning (Refer to Appendix B, *Scanning Multiple Devices*). However, a <wft> without an <sa> can cause the measurement acquisition to stall for single-module scanning applications.

You can force the scanner advanced signal to be generated by inserting <sa> in the scan list. The *Using Advanced Scan List Syntax* section describes how to use <sa>.

Note The functionality of the scanner advanced signal implies that the relays in the switch module have debounced. In further examples, the wait for debounce action associated with each <sa> is not explicitly mentioned.

Scan List

A scan list is made up of one or more scan list entries. Refer to Table 5-6 for scan list examples.

Table 5-6. Scan List Entry Examples

Scan List	Scan List Separated into Individual Entries
`a->b; ~a->b && c->d;`	Entry 1: a->b; Entry 2: ~a->b && c->d;
`a->b; ~a->b && c->d`	Entry 1: a->b; Entry 2: ~a->b && c->d
`a->b`	Entry 1: a->b
Refer to scan list entry definition to see how these entries are interpreted.	

Example 5-1. Basic Scan List Example

The following example uses only the basic scan list syntax.

To measure three voltages using the NI 2503 as a 24–to–1 multiplexer, you can connect the three voltage signals to ch0, ch1, and ch2 of the NI 2503. Then connect the NI 2503 com0 line to the input of a DMM. Finally set the scan list attribute to the string:

```
ch0->com0; ~ch0->com0 &&
ch1->com0; ~ch1->com0 &&
ch2->com0; ~ch2->com0 &&
```

Note New lines are a valid type of white space—they can make a scan list much more readable.

The first line of the scan list string instructs the switch to connect ch0 to com0, wait for the DMM to take a measurement, and then disconnect ch0 from com0 before proceeding to the next line. Similar for the second and third lines.

Notice the `&&` at the end of the third line. This syntax is unneccessary if you will perform this scan list only once. However, if Continuous Scan is set to True, the `&&` is required to ensure that ch2 is disconnected from com0 before repeating the scan list and connecting ch0 to com0.

The scan list string for this simple example is still quite long. The *Using Advanced Scan List Syntax* section describes some syntax features that can shorten this scan list dramatically.

Using Advanced Scan List Syntax

Scan Mode

Scan Mode is not specified in the scan list string itself. Instead, it is an attribute that affects how the scan list string is interpreted.

In the *Using Basic Scan List Syntax* section, all explanations assume that the Scan Mode is set to No Action. With Scan Mode set to No Action, the driver does not insert any disconnect actions.

The *IviSwtch Specification* also allows Scan Mode to be set to Make After Break. NI-SWITCH does not currently support the Make After Break value.

The last value that Scan Mode can be set to is Break Before Make. While in Break Before Make Scan Mode, any connect actions appearing in a semicolon terminated scan list entry are disconnected at the end of that entry (happens after the `<sa>` and `<wft>`). Multiple auto-generated disconnects are separated by connection separators (`&`). If an action separator is needed after the last auto-generated disconnect, then a sequence separator (`&&`) is used.

This functionality is simply for your convenience—the same scan list could be typed out in full strings, such as when Scan Mode is set to No Action.

Table 5-7. Examples of Break Before Make

Break Before Make Scan List	No Action Equivalent
`a->b;`	a->b
`a->b;`	a->b; ~a->b &&
`a->b; c->d;`	a->b; ~a->b && c->d; ~c->d &&
`a->b; c->d`	a->b; ~a->b && c->d
`a->b & c->d; e->f;`	a->b & c->d; ~c->d & ~a->b && e->f; ~e->f &&
`a->b && c->d; e->f;`	a->b && c->d; ~c->d & ~a->b && e->f; ~e->f &&
`a->b;;`	a->b; ~a->b;

Connection Range

Connection range is a shorthand syntax added for your convenience; the syntax is `chanInt:Int->otherChan` or `otherChan->chanInt:Int`. You can use a connection range to specify that many channels be used in place of the range. Either a semicolon or a semicolon modifier[1] must follow the connection range.

Table 5-8. Examples of Connection Range

Scan Mode	Scan List Entry	No Action Expanded Equivalent
No Action	`ch0:2->com0;`	ch0->com0; ch1->com0; ch2->com0;
No Action	`a->b & ch0:2->com0;`	a->b & ch0->com0; ch1->com0; ch2->com0;
Break Before Make	`ch0:2->com0;*`	ch0->com0; ~ch0->com0 && ch1->com0; ~ch1->com0 && ch2->com0; ~ch2->com0 &&

[1] The repeat, scanner advanced, and wait for trigger can serve as semicolon modifiers.

Table 5-8. Examples of Connection Range (Continued)

Scan Mode	Scan List Entry	No Action Expanded Equivalent
Break Before Make	`a->b &` `ch0:2->com0;`	a->b & ch0->com0; ~ch0->com0 && ch1->com0; ~ch1->com0 && ch2->com0; ~ch2->com0 & ~a->b &&

* Notice that this short scan list is equivalent to the lengthy scan list given in Example 5-1.

Repeat

Repeat is most useful for entering several semicolons (used in multimodule scanning), but it can be used in other contexts also. The syntax is `<repeat integer>;`. The word "repeat" is not case sensitive. The trailing semicolon is required.

The text affected by a repeat is defined as the character after the previous semicolon (or the beginning of the scan list if there was no previous semicolon), up to and including the character before the lesser than (`<`) in `<repeat>`. This text is referred to as the repeat body.

The scan list is interpreted the same as a list where the repeat body is typed the specified integer number of times, each time terminated by a semicolon. This rule allows `<repeat>` to serve as a semicolon modifier

Table 5-9. Repeat Examples

Scan List	Expanded Equivalent*
`a->b; <repeat 3>;` `c->d;`	a->b; ;;; c->d;
`ch0:3->com0` `<repeat 6>;`	ch0:3->com0; ch0:3->com0; ch0:3->com0; ch0:3->com0; ch0:3->com0; ch0:3->com0;

* Interpreted according to Scan Mode.

Scanner Advanced

Scanner advanced directs the driver to generate a scanner advanced signal at that point in the scan list. It also allows you to specify the line on which to generate the scanner advanced signal. The syntax for scanner advanced is `<sa optionalLine>`. If *optionalLine* is not present, then the line specified in Scan Advanced Output is used. Refer to Table 5-13 for valid text representations of the possible lines to specify as *optionalLine*.

The line can be the same as that specified in `Scan Advanced Output`, but it need not be.

Note Not all modules are able to use all the functionality of scanner advanced. SCXI-1127/1128 cannot use `<sa>`. PXI can only change the destination to None. SCXI-1129 fully supports `<sa>`.

Scanner Advanced `<sa>` serves as a semicolon modifier when used in the following combinations:

```
<sa>;
<sa> <repeat>;
<sa> <wft>;
<sa> <wft> <repeat>;
```

Otherwise, `<sa>` behaves in the scan list grammar as an action separator. Multiple `<sa>` and/or `<wft>` behave (in the grammar) as a single action separator.

Repeat must be followed by a semicolon. This rule is not changed with the addition of `<sa>`. The combination `<repeat> <sa>;` is illegal. Instead, use the combination `<sa> <repeat>;` Refer to Table 5-10 for Scanner Advanced Examples.

In summary, `<sa>` may act as a modifier for the semicolon, changing the destination of the scanner advanced signal associated with the semicolon. Scanner advanced `<sa>` can also stand alone, causing a scanner advanced signal to be generated where it otherwise would not be generated.

Inserting `<sa>` in a scan list entry means that at least one connect action must appear between the `<sa>` and the scan list entry terminator (which is a semicolon or the end of the scan list) for that terminator to cause another `<sa>` to be generated.

In the following examples, assume that `Scan Advanced Output` is set to ttl0 and that the Scan Mode is set to No Action.

Table 5-10. Scanner Advanced Examples

Scanlist	Meaning
`a->b;`	Connect a to b, then <sa ttl0>, then <wft>
`a->b <sa>;`	Same as above
`a->b <sa ttl5> c->d;`	Connect a to b, then <sa ttl5>, then connect c to d, then <sa ttl0>, then <wft>
`a->b <sa ttl5> c->d`	Connect a to b, then <sa ttl5> then connect c to d, then <sa ttl0>
`c0:3->r0 <sa fc>;`	Connect c0 to r0, then <sa fc>, then <wft> Connect c1 to r0, then <sa fc>, then <wft> Connect c2 to r0, then <sa fc>, then <wft> Connect c3 to r0, then <sa fc>, then <wft>
`c0:3->r0` `<sa fc> <repeat 3>;`	Interpreted the same as: c0:3->r0 <sa fc>; c0:3->r0 <sa fc>; c0:3->r0 <sa fc>;
`~a->b;`	Disconnect a from b, then <wft> Refer to the *Scan List Entry* section for an explanation as to why no <sa> is generated in this case)
`~a->b <sa>;`	Disconnect a from b, then <sa ttl0>, then <wft>

Wait for Trigger

Wait for trigger directs the driver to wait for a trigger at that point in the scan list. It also allows you to specify the line to wait on. The syntax is similar to the scanner advanced syntax: `<wft optionalText>`. If *optionalText* is not present, then the line specified in the Trigger Input attribute is used. Refer to Table 5-13 for valid text representations of the possible lines to program as *optionalText*.

Note Not all modules are able to use all the functionality of wait for trigger.

Note Not all modules are able to use all the functionality of scanner advanced. SCXI-1127/1128 cannot use `<wft>`. PXI can only change the destination to None. SCXI-1129 fully supports `<wft>`.

Wait for trigger `<wft>` serves as a semicolon modifier when used in the following combinations:

```
<wft>;
<wft> <repeat>;
```

Otherwise, `<wft>` behaves in the scan list grammar as an action separator. Multiple `<sa>` and/or `<wft>` entries behave (in the grammar) as a single action separator.

Note Wait for trigger <wft> in the combination `<wft> <sa>;` does not serve as a semicolon modifier. This combination will cause the switch to wait for a trigger, generate a scanner advanced signal, then wait for another trigger.

Similar to `<sa>`, <wft> can be used to modify the trigger associated with a semicolon or it can stand alone, causing the switch to wait for a trigger at a location where it otherwise would not wait.

For the examples in Table 5-11, assume that `Scan Advanced` output is set to ttl5 and that Trigger Input is set to ttl0.

Table 5-11. Wait for Trigger Examples

No Action Scan List	Meaning
`a->b;`	Connect a to b, then <sa ttl5>, then <wft ttl0>
`a->b<wft>;`	Same as above
`a->b<sa><wft>;`	Same as above

Table 5-11. Wait for Trigger Examples (Continued)

No Action Scan List	Meaning
a->b<sa><wft>	Same as above if Scan Mode is No Action. It is interpreted differently than the previous example if Scan Mode is Break Before Make. Refer to Table 5-12.
a->b<sa ttl2> <wft ttl3>;	Connect a to b, then <sa ttl2>, then <wft ttl3>
a->b <wft ttl3>; c->d;	Connect a to b, then <sa ttl5>, then <wft ttl3> Connect c to d, then <sa ttl 5>, then <wft ttl0>

Table 5-12. Wait for Trigger with Break Before Make Examples

Break Before Make Scan List	No Action Equivalent Scan List
a->b<sa><wft>;	a->b <sa> <wft> ~a->b
a->b<sa><wft>	a->b <sa> <wft>
a->b<sa><wft> c->d;	a->b <sa> <wft> c->d <sa> <wft> ~c->d & ~a->b

Table 5-13. Text Representations of Trigger Lines

Text String	Line This Text Specifies	Valid in <sa> commands?	Valid in <wft> commands?
immediate	NISWITCH_VAL_IMMEDIATE	No	Yes
imm	NISWITCH_VAL_IMMEDIATE	No	Yes
sw	NISWITCH_VAL_SW_TRIG_FUNC	No	Yes
none	NISWITCH_VAL_NONE (when used with the <sa> token) NISWITCH_VAL_IMMEDIATE (when used with the <wft> token)	Yes	Yes
external	NISWITCH_VAL_EXTERNAL	Yes	Yes

Table 5-13. Text Representations of Trigger Lines (Continued)

Text String	Line This Text Specifies	Valid in <sa> commands?	Valid in <wft> commands?
ext	NISWITCH_VAL_EXTERNAL	Yes	Yes
ttl0	NISWITCH_VAL_ttl0	Yes	Yes
ttl1	NISWITCH_VAL_ttl1	Yes	Yes
ttl2	NISWITCH_VAL_ttl2	Yes	Yes
ttl3	NISWITCH_VAL_ttl3	Yes	Yes
ttl4	NISWITCH_VAL_ttl4	Yes	Yes
ttl5	NISWITCH_VAL_ttl5	Yes	Yes
ttl6	NISWITCH_VAL_ttl6	Yes	Yes
ttl7	NISWITCH_VAL_ttl7	Yes	Yes
pxi_star	NISWITCH_VAL_PXI_STAR	Yes	Yes
rc	NISWITCH_VAL_REARCONNECTOR	Yes	Yes
fc	NISWITCH_VAL_FRONTCONNECTOR	Yes	Yes

Break

Break (<break>) was specified in the NI-SWITCH 1.5 scan list syntax specification. Some modules can support this functionality; however, NI-SWITCH does not properly support it. Break has been removed from the NI-SWITCH 1.6 scan list syntax specification.

Parsed Scan List

You can read Parsed Scan List to see how the driver parsed the scan list. If an error occurred while parsing the scan list, this attribute contains a parsed version of the scan list up to the error. If no error occurred while parsing, this attribute contains a parsed version of the entire scan list. Read this attribute after writing to Scan List or after calling Configure Scan List.

Configuring the Triggering Options

Configuring the triggering options is very simple for the majority of scanning applications. However, NI-SWITCH 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, *Scanning Multiple Devices*.

One way to trigger, 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.

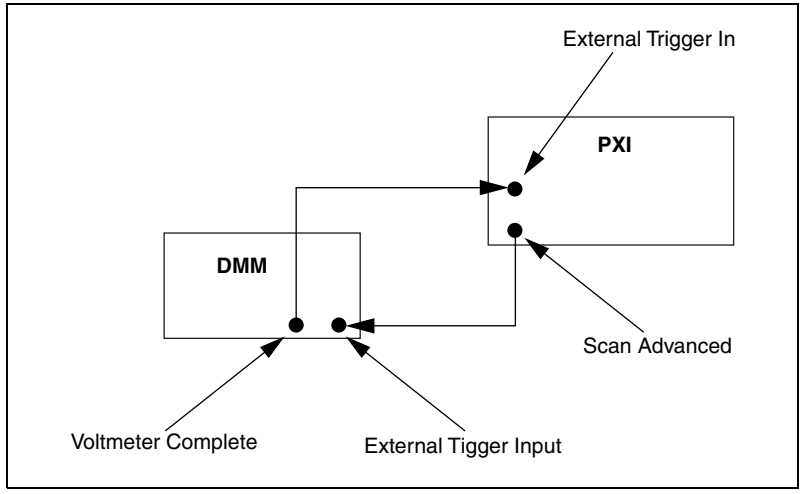

Figure 5-1. Simple Two-Wire Handshake with DMM

Another way to set the **Trigger** parameter is `Software Trigger Function`. This setting tells the switch device that the trigger will not come via a hardware trigger line, but rather from a software command. The program sends the command via `Send Software Trigger`. This setting 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.

Changing the Polarity of the Input Trigger and Scan Advanced

You can change the polarity of the input trigger and the scan advanced trigger by setting Trigger Input Polarity and Scan Advanced Polarity respectively.

Trigger Input Polarity specifies the polarity of the trigger input signal being sent to the switch module. You can specify which edge (rising or falling) of the signal is the active edge. If Trigger Input is specified to be rising edge, then the switch module triggers on the rising edge of the input trigger. Refer to the *NI-SWITCH C Reference Help* or *NI-SWITCH VI Reference Help* for a full description and the valid values for this attribute.

Scan Advanced Polarity effects the polarity of the scan advanced signal that the switch module sends. If Scan Advanced Polarity is specified to be rising edge, then the switch module generates positive scan advanced pulses. Refer to the *NI-SWITCH C Reference Help* or *NI-SWITCH VI Reference Help* for a full description and the valid values of this attribute.

Using Scan Delay

The switch device generates a scan advanced trigger after all the switches have settled. This action alerts the measurement device that it can now take a measurement. The switch devices themselves are designed to wait a specified amount of time (Refer to Settling Time in the *NI-SWITCH C Reference Help*) to ensure the switch has settled. However, due to capacitance in the system, the switch 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 in `Configure Scan Trigger`.

Combining Scanning and Routing Functions

NI-SWITCH does not support the use of routing functions, such as `Connect` and `Disconnect`, while a scan is in progress. However, your application may need to use the switch routing functions in some parts of your code, while other sections may need to use scanning functions. An NI-SWITCH application might need to know what state the switch device needs to be in before a scan can be initiated, and what state the switch device is left in when a scan completes or is aborted.

You can use `Disconnect All` after a scan or after a series of routing functions to disconnect all existing paths on the switch device. Most applications should call `Disconnect All` between routing and scanning functions to put the switch device in a known state.

Scan Operations

In *Example 3-5. Scanning*, you saw that performing a scan takes very little coding. `Scan` uses the scan list as its main parameter, parses the scan list, programs the hardware, and initiates the scan before returning. Also, it does not wait for the scan to complete before it returns.

You can use `Is Scanning` or `Wait For Scan Complete` to determine if the device is currently in Scan Mode. However, if the continuous mode (via `Set Continuous Mode`) is set to **True**, the switch device has no way of knowing the scan is complete, so neither of these operations tells the information you need. Instead, track the state of the scan by monitoring the measurement device since the measurement device has the actual count of measurements to be made.

Note `Wait for Scan Complete` is only valid for PXI and cannot be used with SCXI.

Example 5-2. Scan Operations and Programming Example

Example 5-2 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.

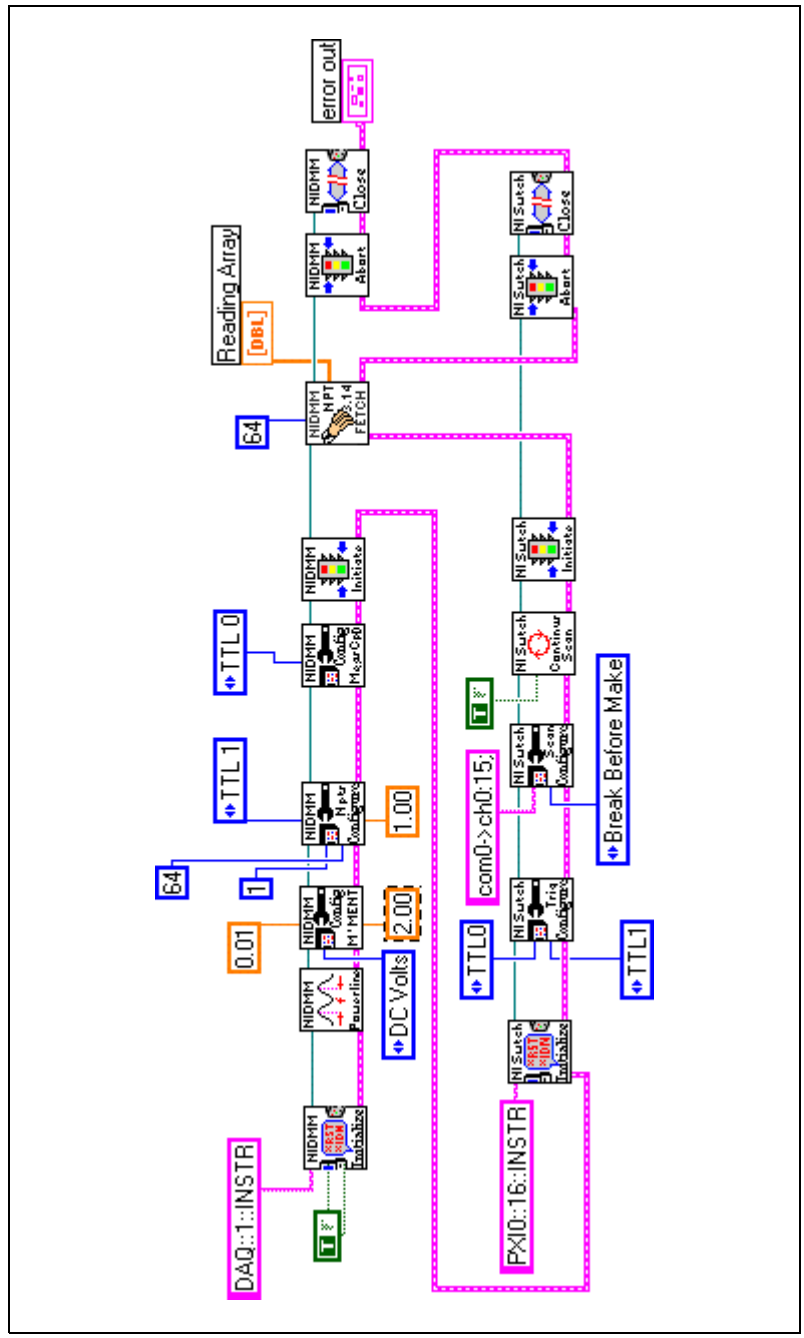

Figure 5-2. Scanning the NI 2503 Using the NI 4060 DMM Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViSession instr           = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 10.00;
   ViReal64  resolution      = 0.01;
   ViInt32   numOfMeas       = 64;
   ViReal64  measurements[64];
   ViInt32   numpts;
/*- NI-SWITCH configurations-----------------------------------------*/
   checkErr(niSwitch_init ("PXI0::16::INSTR", VI_TRUE,  VI_TRUE, &instr));
  checkErr(niDMM_init(resourceName, idQuery, reset, &vi));

   checkErr(niSwitch_ConfigureScanlist (instr, "com0->ch0:15;",
            NISWITCH_VAL_BREAK_BEFORE_MAKE));

   checkErr(niSwitch_ConfigureScanTrigger (instr, 0.0,
            NISWITCH_VAL_ttl0,
            NISWITCH_VAL_ttl1));

   checkErr(niSwitch_SetContinuousScan (instr, 1));
   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement (vi, function, range, resolution));
   /*- Configure Powerline frequency-----------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency (vi, powerlineFreq));
   /*- Configure a multipoint acquisition------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, NIDMM_VAL_ttl1,
                    0.5));
   /*-ConfigureMeasurementCompleteDestination-----------------------*/
   checkErr(niDMM_ConfigureMeasurementComplete (vi, NIDMM_VAL_ttl0,
                    NIDMM_VAL_POS));
   /*- Initiate switch and DMM-----------------------------------------*/
   checkErr(niDMM_Initiate (vi));
   checkErr(niSwitch_InitiateScan (instr));
   /*- Read 64 measurementswith DMM------------------------------------*/
```

```
    checkErr(niDMM_FetchMultiPoint (vi, NIDMM_VAL_TIME_LIMIT_AUTO, 64,
                     measurements, &numpts));

  /*- Abort and Close DMM and Switch--------------------------------------*/
    checkErr(niDMM_close (vi));
    checkErr(niSwitch_AbortScan (instr));
    checkErr(niSwitch_close (instr));

Error:
    if (vi)
       niDMM_close(vi);

    if (error < VI_SUCCESS)
       messageHandler(error);
    }
```

6

Using NI-SWITCH with SCXI

Several conventions have been added to NI-SWITCH to add support for SCXI. These conventions are SCXI Instrument Descriptors, SCXI Scan Lists, and Trigger Naming.

For your switch system to operate correctly, you need to properly configure your hardware. Refer to your hardware user manual for information on configuration and self-test. For information about topologies for your switch module, refer to the appropriate section that follows.

Note The SCXI-1127, 1128, and 1129 modules support scanning (refer to Chapter *5, Scanning*). All other SCXI modules do not support scanning.

Switch Topologies for SCXI-1127 and SCXI-1128

This section describes how NI-SWITCH supports the SCXI-1127 and SCXI-1128.

Three instrument descriptors have been added to describe various topologies in which the SCXI-1127 and SCXI-1128 can operate. These topologies are *Scanner*, *Matrix*, and *Independent*. In Scanner Mode, you can combine several SCXI-1127/1128s into one scanner from a software point of view. This mode simplifies configuration and programming. In Matrix mode, you can use your SCXI-1127/1128 as a matrix, directly controlling the connection of row and columns of a matrix. In Independent mode, you can individually control any switch. This mode is supported on the SCXI-1127, SCXI-1128, SCXI-1160, SCXI-1161, and SCXI-1163R.

Two terminal blocks, SCXI-1331 and SCXI-1332, can be used with the SCXI-1127/1128. The SCXI-1331 is designed for scanner operation. The SCXI-1332 is specifically designed for matrix operation.

For correct operation of your system, confirm that your SCXI-1127/1128 has been properly configured. Refer to the *SCXI-1127/1128 User Manual* for the detailed configuration procedure for the SCXI-1127/1128 module.

After configuring the module, complete the following procedure to select the appropriate terminal block:

1. Double-click the **Measurement & Automation Explorer** icon on your computer desktop.
2. Expand **Devices and Interfaces**.
3. Right-click SCXI-1127 or SCXI-1128 and select **Properties**.
4. Click the **Accessory** tab and select the proper terminal block.

The following sections describe these three topologies and the conventions used for each type.

Scanner Mode

When programming, the general form for the instrument descriptor is as follows:

`SCXI`*`n`*`::`*`m1,m2,m3`*`::SCANNER`

where

n = chassis ID

m1,m2,m3 = module slot number

For example, to create a scanner out of three SCXI-1127/1128s in slots 5, 6, and 8 of chassis 2, the instrument descriptor would be as follows:

```
SCXI2::5,6,8::SCANNER
```

For just one SCXI-1127/1128, in slot 4 of chassis 1, the instrument descriptor would be as follows:

```
SCXI1::4::SCANNER
```

To make all of chassis 1 a scanner, a shortcut you can use is as follows:

```
SCXI1::SCANNER
```

Scans cannot span multiple chassis. A unique session must be created for each chassis.

The scanner configuration uses the scanning function calls. These calls require scan lists. A scan list is a string that specifies the channel connections for scanning. The scan list is comprised of channel names that are separated with special characters. These special characters determine the operation the scanner performs on the channels when it executes this scan list. For proper scanning operation on SCXI-1127/1128, the

Continuous Scan parameter in `Set Continuous Scan` must be set to True. Also, you must call `Abort Scan` must be called to stop the scan.

Before building a scan list, you should understand the SCXI channel string nomenclature.

Single Channel

The general form is as follows:

```
sc<chassis ID number>!md<module slot number>!ch<channel number>
```

For example, channel 3, on module 4, in chassis 2 would be specified as follows:

```
sc2!md4!ch30
```

To create a path between two channels, use `->` (a dash followed by a greater than sign) between the two channel names.

For example, to scan the channel specified above using the com0 bus, the syntax would be as follows:

```
sc2!md4!ch3->com0;
```

Note A semicolon is used to indicate that the SCXI-1127/1128 should wait for a trigger before proceeding to the next entry in the scan list.

Multiple Sequential Channels

To scan multiple channels in a scan list, concatenate these paths together.

For example, to scan channel 3, 4, and 5 on module 12 in chassis 1 over the com0 bus, the syntax would be as follows:

```
sc1!md12!ch3->com0;sc1!md12!ch4->com0;sc1!md12!ch5->com0;
```

This example performs the following actions:

1. Select channel 3.
2. Wait for a trigger.
3. Select channel 4.
4. Wait for a trigger.
5. Select channel 5.

A simpler way to group channels in a scan list is by using the colon. The colon is used to delineate between a start channel and an end channel.

For example, you can use the colon in the example above. That string could be rewritten as follows:

```
sc1!md12!ch3:5->com0;
```

This command scans and triggers exactly like the previous example. There is no limitation on the order of the channel sequence on the SCXI-1127/1128. To scan the channels in the opposite order, enter the scan list as follows:

```
sc1!md12!ch5:3->com0;
```

In fact, the SCXI-1127/1128 can scan channels in any order.

Multiple Random Channels

For example, to scan channel 8 on module 2, then channel 4 on module 4, and channel 12 on module 3 in chassis 1 over the com0 bus the syntax would be as follows:

```
sc1!md2!ch8->com0;

sc1!md4!ch4->com0;

sc1!md3!ch12->com0;
```

This example performs the following actions:

1. Select channel 8 (module 2).
2. Wait for a trigger.
3. Select channel 4 (module 4).
4. Wait for a trigger.
5. Select channel 12 (module 3).

Cold-Junction Temperature Sensor Channel

The SCXI-1331 terminal block contains a cold-junction temperature sensor. This sensor connects to a special channel on the SCXI-1127/1128 dedicated to measuring the ambient temperature of the terminal block. This channel is used when measuring thermocouples. This channel is always scanned as a 2-wire channel. You can include the cold-junction temperature sensor channel at any position in the list with any number of repetitions by indicating it with the name *cjtemp*.

For example, to scan the cold-junction temperature sensor, and channels 3, 8, and 5 on module 12 in chassis 1 over the com0 bus, the syntax would be as follows:

```
sc1!md12!cjtemp->com0;
sc1!md12!ch3->com0;
sc1!md12!ch8->com0;
sc1!md12!ch5->com0;
```

Analog Bus Configuration

The analog bus channels are automatically connected to the high-voltage analog backplane on the modules you include in the scanner instrument descriptor at the time you initiate the scan. In a multimodule scan, this feature connects the output of the SCXI-1127/1128 to the high-voltage analog backplane, which is generally connected to a DMM.

In some instances, it you may want to scan a module without the output connected to the analog backplane. For example, you can to route the output of the multiplexer to the output terminal of the terminal block, with an access from the front of the chassis, instead of routing them to the high-voltage analog backplane. You can achieve this action by calling `Initialize With Options` with the following configuration string:

```
DriverSetup = SCXI-1127 MUX manual AB
```

This configuration string opens a session to a scanner, but leaves the analog buses open. To close the analog bus you must call `Connect` and route the common bus to the analog bus. For example, to close the ab0 switch, call `Connect` with the channel 1 parameter set to com0 and the channel 2 parameter set to ab0. For more details, refer to the *SCXI-1127/1128 User Manual*.

You can also close ab2 in a similar manner. You would call `Connect` with the channel 1 parameter set to com0 and the channel 2 parameter set to ab2. This setting is important if you are in a 4-wire configuration. If you are using a National Instruments DMM with the SCXI-1127/1128, keep in mind that the DMM uses both the ab0 and ab2 buses for making measurements. The ab2 bus is used as the sense lines for 4-wire resistance measurements and for current measurements.

Route Functions

You can also route signals while in scanner mode. The route functions allow you to connect/disconnect from/to one point to/from another point. For example, to connect channel 0 to analog bus 0 you call Connect with the channel 1 parameter set to ch0 and the channel 2 parameter set to com0. Then call `Connect Channels` with the channel 1 parameter set to com0 and the channel 2 parameter to ab0. A list of valid channels for the SCXI-1127/1128 is listed in the following sections.

Analog Bus Channel <ab0...ab3>

Analog bus channels <ab0...ab3> are the four signals that comprise the SCXI high-voltage analog bus on the SCXI-1127/1128.

Common Bus Channel <com0>

Common bus channel is the internal bus on the scanners. On the SCXI-1127/1128, the input channels can be connected to the common bus. The common bus can also be connected to the analog bus channels by the switch ab0.

Input Channels <ch0…ch63>

The SCXI-1127/1128 has 64 1-wire input channels. You can configure channels <ch0…ch31> as 2-wire inputs, and you can configure channels <ch0…ch15> as 4-wire inputs.

Cold-Junction Temperature Sensor (cjtemp)

You can access the cold-junction temperature sensor channel on the SCXI-1331 terminal block through the SCXI-1127/1128. To read from this channel call Connect with the channel 1 parameter set to cjtemp and the channel 2 parameter to com0. Then call Connect with the channel 1 parameter set to com0 and the channel 2 parameter to ab0. The output of the cold-junction temperature sensor will now be present on analog bus 0.

Configuring the Input Channels

In the scanner mode, you can configure the input channels on a per-channel basis by choosing between 1-wire, 2-wire, or 4-wire mode. Do this configuration through MAX. To configure the SCXI-1127/1128, follow these steps:

1. Double-click the **Measurement & Automation Explorer** icon on your computer desktop.
2. Expand **Devices and Interfaces**.

3. Right click SCXI-1127/1128 and select **Properties**.
4. Click the **Channels** tab, and set the proper input mode configuration for the channels you are using.

Matrix Mode

When programming, the general form for the instrument descriptor is as follows:

`SCXI`*`n`*`::`*`m`*`::MATRIX`

where

n = chassis ID

m = module slot number

For example, to create a matrix out of a SCXI-1127/1128 that you have in slot 12 in chassis 1, the instrument descriptor would be as follows:

```
SCXI1::12::MATRIX
```

In this version, a unique session must be created for each matrix module. The matrix configuration uses the route functions. When a SCXI-1127/1128 is configured as a matrix, it creates a 4×8 (4 rows by 8 columns) matrix.

Note The SCXI-1127/1128, when configured as a matrix, must use the SCXI-1332 terminal block and must have the accessory field set appropriately in MAX; otherwise, an error will result during program execution.

You can expand the four columns by connecting the rows together through the high-voltage analog backplane. Both columns and rows can be expanded through the SCXI-1332 terminal block using the Matrix Expansion Cable from National Instruments. Unlike the scanner topology that uses a channel naming convention, the matrix uses a column/row naming convention.

For example, to connect the row 0 to column 3, call `Connect` with the channel 1 parameter set to r0 and the channel 2 parameter set to c3. The row names are r0, r1, r2, and r3. The column names are c0, c1, c2, c3, c4, c5, c6, and c7.

To use the analog bus and the high-voltage backplane for expansion, call `Connect`. For example, to connect r0 to ab0, call `Connect` with the channel 1 parameter set to r0 and the channel 2 parameter set to ab0. To

connect all of the rows, close the rest of the switches in a similar manner (connect r1 to ab1, connect r2 to ab2, and connect r3 to ab3).

Independent Mode

The general form is as follows:

`SCXI`*`n`*`::`*`m`*`::INDEP`

where

n = chassis ID

m = module slot number

For example, to control a SCXI-1160 that is in slot 12 in chassis 1, the instrument descriptor would be as follows:

`SCXI1::12::INDEP`

A unique session must be created for each module. The independent configuration uses the low-level functions such as `Single Switch`. The switch names used in these low-level functions refer to physical switches on the module.

Before using the low-level functions, it is important to understand the switch naming conventions, shown in the *SCXI-1127/1128 User Manual*.

The SCXI-1127/1128 is comprised of switches that are opened or closed based on the configuration of the software. These switches and their names are listed in the following sections.

Analog Bus Switches <ab0…ab3>

These switches connect/disconnect the SCXI-1127/1128 internal common bus to/from the high-voltage analog backplane.

Channel Switches <ch0…ch31>

These switches connect/disconnect the input signals to/from the SCXI-1127/1128 internal common bus.

Bank Connect Switches <bc01…bc23>

The input of the SCXI-1127/1128 consists of four 8 to 1 multiplexers. These multiplexers can be connected together by closing the bank connect switches. Specifically, bc01 connects bank 0 to bank 1, bc02 connects bank 0 to bank 2, and bc23 connects bank 2 to bank 3.

Cold Junction Temperature Sensor (cjtemp)

You can connect the cold junction temperature sensor on the SCXI-1331 terminal block through the SCXI-1127/1128. Closing this switch will connect the cold junction temperature sensor to the internal common bus on the SCXI-1127/1128.

Triggering

You can use trigger functions when the SCXI-1127/1128 is in Scanner Mode and when in conjunction with other scan functions. New trigger line options have been added to `Configure Scan Trigger`.

Note The SCXI-1127/1128 does not support the **Scan Delay** parameter in this function.

This function includes the parameters **Trigger Input** and **Scan Advanced Output**.

Trigger Input

Pass the trigger source you want the SCXI-1127/1128 to use to advance to the next entry in the scan list. The driver uses this value to set the Trigger Input attribute.

Note The module that receives the trigger must be part of the scanner instrument descriptor. Although, this module does not have to be in a scan list.

The legal values for the **Trigger Input** parameter are as follows:

- Immediate—Not valid on SCXI-1127/1128
- External—Not valid on SCXI-1127/1128
- TTL(0)—This variable maps the TTL0 trigger signal to the TRIG0 line on the SCXI backplane. For example, you could use this variable in the PXI-1010 chassis. The module in PXI slot 8 of the PXI-1010 chassis would route its trigger over the TTL(0) bus to trigger the SCXI-1127/1128. For example, a digital multimeter in PXI slot 8 of the PXI-1010 chassis would trigger the SCXI-1127/1128 over the TTL(0) trigger. For other chassis, consult your chassis manual for support of this trigger.
- TTL <1...7>—Not valid on SCXI-1127/1128
- ECL0—Not valid on SCXI
- ECL1—Not valid on SCXI
- PXI Star—Not valid on SCXI

- Software Trigger—The switch module waits until you call `Send Software Trigger`.
- Rear Connector—The switch waits until it receives a trigger on the rear connector before processing the next entry in the scan list. This variable is valid for SCXI scanners that consist of a single module. If more than one module is used, you must specify which slot is receiving the trigger by selecting the Rear Connector of Module <1...12> setting instead.
- Front Connector—The switch waits until it receives a trigger on the front connector before processing the next entry in the scan list. When using SCXI scanners, this variable is valid for scanners that consist of a single module. If more than one module is used, you must specify which slot is receiving the trigger input by using the Front Connector of Module <1...12> instead.
- Rear Connector of Module <1...12>—The switch waits until it receives a trigger on the rear connector of slot <1...12> before processing the next entry in the scan list. This variable specifies, in a multimodule SCXI-1127/1128 scanner, which module is receiving the trigger input from its rear connector; all other modules that are part of the scanner receive their trigger input from TRIG0. TRIG0 routing is done implicitly by the software. Confirm that the module that you selected is configured in MAX as the cabled module.
- Front Connector of Module <1...12>—The switch waits until it receives a trigger on the front connector of slot <1...12> before processing the next entry in the scan list. This variable specifies, in a multimodule SCXI-1127/1128 scanner, which module is receiving the trigger input from its front connector; all other modules that are part of the scanner receive their trigger input from TRIG0. TRIG0 routing is done implicitly by the software.

Scan Advanced

The scan advanced trigger is used to indicate that the SCXI-1127/1128 has switched to the next channel and that the switch has settled.

Note Scanner advanced and scan advanced are different names for the same trigger.

After the SCXI-1127/1128 processes each entry in the scan list, it waits 9 ms and then asserts a trigger on the line you specify with this parameter.

Note The **Scan Delay** parameter is not supported on the SCXI-1127/1128.

The legal values for the **Scan Advanced** parameter are as follows:

- None—The switch module does not produce a scan advanced output trigger. Use this setting when you are using a National Instruments DMM as the SCXI controller and measurement device.
- External—Not valid on SCXI-1127/1128
- TTL <0, 1>—Not valid on SCXI-1127/1128
- TTL <2>—SCXI TTL <2>. This variable maps the SCXI-1127/1128 scanner advanced trigger signal to the TRIG2 line on the SCXI backplane. For example, you could use this variable in a SCXI-2000 chassis. One SCXI-1127/1128 could route its scanner advanced signal over the SCXI backplane so that another module could route this signal to its front connector. For other chassis, consult your chassis manual for support of this trigger.
- TTL <3…7>—Not valid on SCXI-1127/1128
- ECL0—Not valid on SCXI
- ECL1—Not valid on SCXI
- PXI_STAR—Not valid for SCXI
- Rear Connector—Not valid for SCXI
- Front Connector—This variable indicates that the switch module sends its Scanner Advanced output to the front connector. When using SCXI switches as scanners, all the modules that are part of the scanner send their Scanner Advanced output to their respective front connectors.
- Rear Connector of module <1...12>—Not valid for SCXI
- Front Connector of module <1...12>—When using SCXI switches as scanners, this variable indicates which module is bussing the Scanner Advanced output to its front connector for access by other instruments, all other modules in the Scanner send their Scanner Advanced to this module via TRIG2. Consult your chassis manual to find out if TRIG2 is supported.

Note For SCXI-1127/1128, Trigger Input Polarity and Scan Advanced Polarity cannot be set to different values. Changing Trigger Input Polarity will change Scan Advanced Polarity to the same value and vice versa.

Note When performing a scan on SCXI-1127/1128 in a PXI-1010 or PXI-1011 chassis through the backplane, Trigger Input attribute should be set to TTL0.

Switch Topologies for the SCXI-1160, SCXI-1161, and SCXI-1163R

This section describes how NI-SWITCH supports the SCXI-1160, SCXI-1161, and SCXI-1163R.

SCXI-1160

The SCXI-1160 module can be operated in two modes: independent mode or instrument mode.

When you configure the SCXI-1160 in INDEP (independent) mode, the SCXI-1160 contains 16 SPDT relays. These channels are referred to as <ch0...ch15>. You can control the SCXI-1160 through low-level function calls such as `Single Switch`.

For example, to close relay ch2 on the SCXI-1160, you will call `niSwitch_SingleSwitchControl(vi, "ch2", 1)`. This action connects the NO (normally opened) terminal of channel 2 to the COM terminal of channel 2, and disconnects the NC (normally closed) terminal of channel 2 from the COM terminal.

When you configure the SCXI-1160 as INSTR, these channels are referred to as follows:

```
NC0, NO0, COM0
NC1, NO1, COM1
.....
NC15, NO15, COM15
```

In this configuration, you can control the SCXI-1160 through the function call `Connect`.

For example, to connect the NO terminal of channel 2 to the COM terminal of channel 2, and to disconnect the NC terminal of channel 2 from the com terminal, call `niSwitch_Connect(vi, "NO2", "COM2")`. If you now want to connect NC2 to com2, you need to first disconnect the existing connection. The sequence of calls for this task is as follows:

```
niSwitch_Disconnect(vi, "NO2", "COM2")
niSwitch_Connect(vi, "NC2", "COM2")
```

Note In C programming, `niSwitch_Disconnect(vi, "NO2", "COM2")` does not activate the relay until the `niSwitch_Connect(vi, "NC2", "COM2")` is executed.

SCXI-1161

The SCXI-1161 module can be operated in two modes: independent mode or instrument mode.

When you configure the SCXI-1161 in INDEP mode, the SCXI-1161 contains eight SPDT relays. These channels are referred to as <ch0...ch7>. You can control the The SCXI-1161 through low-level function calls such as `Single Switch`.

For example, to close relay ch2 on the SCXI-1161, call `niSwitch_SingleSwitchControl(vi, "ch2", 1)`. This action connects the NO terminal of channel 2 to the COM terminal of channel 2, and disconnects the NC terminal of channel 2 from the com terminal.

When you configure the SCXI-1161 as INSTR, these channels are referred to as follows:

```
NC0, NO0, COM0
NC1, NO1, COM1
.....
NC7, NO7, COM7
```

In this configuration, you can control the SCXI-1161 through the function call `Connect`.

For example, to connect the NO terminal of channel 2 to the com terminal of channel 2, and to disconnect the NC terminal of channel 2 from the com terminal, call `niSwitch_Connect (vi, "NO2", "COM2")`. If you now want to connect NC2 to com2, you need to first disconnect the existing connection. The sequence of calls for this task is as follows:

```
niSwitch_Disconnect (vi, "NO2", "COM2")
niSwitch_Connect (vi, "NC2", "COM2")
```

Note `niSwitch_Disconnect (vi, "NO2", "COM2")` does not activate the relay until the `niSwitch_Connect (vi, "NC2", "COM2")` is executed.

SCXI-1163R

The SCXI-1163R can only be operated in the INDEP (independent) mode. The SCXI-1163R contains eight banks of four input channel switches connected to a common channel. These input channels are referred to as <ch0…ch31>. The eight common channels are referred to as <com0…com7>. Because the SCXI-1163R is comprised of eight banks of

four switches each, you can only connect to the common channel that is in your bank. The banks are organized as such:

```
ch0, ch1, ch2, ch3, com0
ch4, ch5, ch6, ch7, com1
ch8, ch9, ch10, ch11, com2
ch12, ch13, ch14, ch15, com3
ch16, ch17, ch18, ch19, com4
ch20, ch21, ch22, ch23, com5
ch24, ch25, ch26, ch27, com6
ch28, ch29, ch30, ch31, com7
```

For example, you can connect ch8 to com2; however, you cannot connect ch8 to com6.

The SCXI-1163R can be controlled through low-level function calls such as `Single Switch`. For example, to close relay 2 on the SCXI-1163R you call `Single Switch` with the action name set to close and the switch name parameter set to ch2.

The SCXI-1163R can also be controlled through route functions calls such as `Connect`. For example, to connect channel 16 to common 4, you call `Connect` with the channel 1 parameter set to ch16 and the channel 2 parameter set to com4.

Switch Topologies for the SCXI-1190, SCXI-1191, and SCXI-1192

This section describes how NI-SWITCH supports the SCXI-1190, SCXI-1191, and SCXI-1192.

SCXI-1190, SCXI-1191

The SCXI-1190/1191 are general-purpose, quad 4-to-1, high-bandwidth multiplexers. The SCXI-1190 uses single-pole double-throw high-bandwidth relays capable of switching signals from DC to 1.3 GHz. Functionally identical, the SCXI-1191 has a bandwidth of DC to 4 GHz. The SCXI-1190/1191 modules can be operated in two modes: independent mode or instrument mode.

When you configure the SCXI-1190/1191 in INDEP mode, the SCXI-1190/1191 contains 16 relays. These channels are referred to as follows:

```
sw0A, sw1A, sw2A, sw3A
sw0B, sw1B, sw2B, sw3B
sw0C, sw1C, sw2C, sw3C
sw0D, sw1D, sw2D, sw3D
```

The SCXI-1190/1191 can be controlled through low-level function calls such as `Single Switch`.

For example, to close relay sw1A on the SCXI-1190 (meaning connection between swA1 and comA), call `niSwitch_SingleSwitchControl (vi, "sw1A", 1)`.

When you configure the SCXI-1190/1191 as INSTR, the SCXI-1190/1191 contains 16 relays. When configured as INSTR, these channels are referred to as follows:

```
ch0A, ch1A, ch2A, ch3A
ch0B, ch1B, ch2B, ch3B
ch0C, ch1C, ch2C, ch3C
ch0D, ch1D, ch2D, ch3D
```

In this configuration, the SCXI-1190/1191 can be controlled through the function call `Connect`.

For example, to connect relay ch1A to comA, call `niSwitch_Connect (vi, "ch1A", "comA")`. If you now want to connect ch2A to comA, you need to first disconnect the existing connection. The sequence of calls for this task is as follows:

```
niSwitch_Disconnect (vi, "ch1A", "comA")
niSwitch_Connect (vi, "ch2A", "comA")
```

Note `niSwitch_Disconnect (vi, "ch1A", "comA")` does not activate the relay until the `niSwitch_Connect (vi, "ch2A", "comA")` is executed. Also remember that you must always have a closed connection inside a bank.

SCXI-1192

The SCXI-1192 module can be operated in two modes: independent mode or instrument mode.

When you configure the SCXI-1192 in INDEP mode, the SCXI-1192 contains eight SPDT relays. These channels are referred to as <ch0...ch7>. You can control the The SCXI-1192 through low-level function calls such as `Single Switch`.

For example, to close relay ch2 on the SCXI-1192, call `niSwitch_SingleSwitchControl(vi, "ch2", 1)`. This action connects the NO terminal of channel 2 to the com terminal of channel 2, and disconnects the NC terminal of channel 2 from the com terminal.

When you configure the SCXI-1192 as INSTR, these channels are referred to as follows:

```
NC0, NO0, COM0
NC1, NO1, COM1
.....
NC7, NO7, COM7
```

In this configuration, you can control the SCXI-1192 through the function call `Connect`.

For example, to connect the NO terminal of channel 2 to the com terminal of channel 2, and to disconnect the NC terminal of channel 2 from the com terminal, call `niSwitch_Connect (vi, "NO2", "COM2")`. If you then want to connect NC2 to com2, you need to first disconnect the existing connection. The sequence of calls for this task is as follows:

```
niSwitch_Disconnect (vi, "NO2", "COM2")
niSwitch_Connect (vi, "NC2", "COM2")
```

Note `niSwitch_Disconnect (vi, "NO2", "COM2")` does not activate the relay until the `niSwitch_Connect (vi, "NC2", "COM2")` is executed.

Switch Topologies for the SCXI-1129

The SCXI-1129 is a 256 crosspoint high-density matrix module. The SCXI-1129 can operate as four 4 × 16, 2-wire matrixes; two 4 × 32, 2-wire matrixes; a 4 × 64, 2-wire matrix; two 8 × 16, 2-wire matrixes; a 8 × 32, 2-wire matrix; or a 16 × 16, 2-wire matrix. The following instrument descriptors have been added to describe various topologies in which the SCXI-1129 can operate:

- MATRIX_4x16—four 4 × 16, 2-wire matrixes
- MATRIX_4x32—two 4 × 32, 2-wire matrixes
- MATRIX_4x64—one 4 × 64, 2-wire matrix
- MATRIX_8x16—two 8 × 16, 2-wire matrixes
- MATRIX_8x32—one 8 × 32, 2-wire matrix
- MATRIX_16x16—one 16 × 16, 2-wire matrix

For example, to configure the SCXI-1129 in the 4 × 64 topology, the instrument descriptor is:

`SCXI`*`n`*`::`*`m`*`::MATRIX_4x64`

where

n = chassis ID

m = module slot number

You can also use MATRIX as an instrument descriptor. If you use MATRIX, NI-SWITCH picks a topology based on the terminal block you configured in MAX. Refer to Table 6-1.

Table 6-1. Mapped Topologies in MAX

Terminal Block	Matrix Topology
SCXI-1333	MATRIX_4x16
SCXI-1334	MATRIX_4x64
SCXI-1335	MATRIX_8x32
SCXI-1336	MATRIX_16x16
SCXI-1337	MATRIX_8x16
SCXI-1339	MATRIX_4x32

Note If you specify MATRIX as the instrument descriptor and do not configure a specific terminal block in MAX, then the SCXI-1129 defaults to the MATRIX_4x16 topology.

Any of the instrument descriptors allows you to perform the following operations on the SCXI-1129:

- Route signals with the connect/disconnect functions
- Scan a list of channels
- Manually control individual switches

The analog bus relays are *not* automatically connected when configuring the SCXI-1129 in any of the above mentioned topology.

Routing Signals

Using the SCXI-1129, you can route signals in any one of six configurations: MATRIX_4x16, MATRIX_4x32, MATRIX_4x64, MATRIX_8x16, MATRIX_8x32, or MATRIX_16x16.

For example, calling `Connect` with channel 1 set to B4R3 and channel 2 set to B4C3 routes signals from row 3 of bank 4 to column 3 of bank 4. For more information about matrix topologies, refer to the *SCXI-1129 User Manual*.

Scanning a List of Channels

You can scan through a list of channels on the SCXI-1129 by specifying the triggering information and a scan list. The instrument descriptor does not change when scanning on the SCXI-1129. For example, you can use MATRIX_4x16 to route channels and scan on the SCXI-1129. Refer to Chapter 5, *Scanning*, for a description of scanning in general and the scan list syntax.

Analog Bus Configuration for Scanning

The analog bus channels are not automatically connected to the HVAB on the SCXI-1129. Connecting the analog bus channels enables a DMM that is cabled to the HVAB to take measurements. To close the analog bus channels, you can specify the analog bus as part of the scan list. For example, a scan list entry of `r0->com0 && com0->ab0;` connects row 0 to analog bus 0. You can also close the analog bus channels by calling `Connect` with the channel 1 set to com0 and channel 2 set to ab0.

Note If the analog bus relays are closed during a scan, they remain closed until `Abort Scan` is called.

Manual Control of Switches

You can use `SingleSwitchControl` to individually control the switches on the SCXI-1129. Refer to the *SCXI-1129 User Manual* for the switch names used in this function.

If you are doing manual switch control, use MATRIX_4×16 for the instrument descriptor.

Scanning a Non-Cabled SCXI Module

A cabled SCXI module has the measurement device, a DMM for example, directly connected to it. To scan a module indirectly connected to a measurement device (the non-cabled module), the cabled module has to route the trigger signals from the measurement device to the non-cabled SCXI module. The cabled module routes the input triggers to one of the trigger lines (ttl lines) on the SCXI backplane and the non-cabled module picks up the input triggers from these ttl lines. You can select the ttl line that the cabled module routes the triggers to by setting the Cabled Module Trigger Bus attribute.

The Cabled Module Scan Advanced Bus can also be used while scanning a non cabled SCXI module. This attribute selects the ttl line that the non-cabled module sends its scan advanced to. The scan advanced is then routed from the ttl line to the measurement device by the cabled module.

NI-SWITCH checks the Trigger Input attribute, Scan Advanced attribute and Measurement & Automation Explorer (MAX) to determine the cabled module.

Example 6-1 Scanning

This example illustrates how the Cabled Module Trig Bus and Cabled Module Scan Advanced Bus work.

If your SCXI chassis has an SCXI-1129 in slot 3 and another SCXI-1129 in slot 4 connected to a DMM through the Front Connector.

If you want to perform handshaking with the non-cabled SCXI-1129 (in slot 3), you have to set the Trigger Input and Scan Advanced attribute (for the module in slot 3) to be the Front Connector of Module 4. If you set the Cabled Module Trigger Bus to ttl5 and Scan Advanced Trig Bus to be ttl6, the cabled SCXI-1129 in slot 4 routes the input triggers from the DMM to ttl5 and routes the scanner advanced from ttl6 to the DMM. By setting

these two attributes, the SCXI-1129 in slot 3 receives its triggers from ttl5 and sends scanner advanced to ttl6.

Note Not all SCXI modules that support scanning can route to all trigger lines on the SCXI backplane. Currently only SCXI-1129 can route to all trigger lines. SCXI-1127 and SCXI-1128 can route input triggers to ttl0 and route scanner advanced from ttl2. Not all triggers lines are supported in all chassis. Make sure that the correct chassis is chosen for your particular application.

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 3-1

```
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 3-2

```
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 3-3

```
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 3-4

```
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 3-5

```
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;",NISWITCH_VAL_SWITCH_INITIATED);

        /* 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 dmmRange as ViReal64         ' 0.2 Volt Range
        Dim triggerDelay as ViReal64     ' No Trigger Delay
        Dim handInit as ViBoolean        ' DMM Initiates Acquistion
        Dim numOfMeas as ViInt32         ' Number of Points to Take
        Dim measurements as ViReal64 * 24' Array for Data

        Dim status as ViStatus

        REM Initialize DMM variables

        triggerDelay = 0.0
        handInit = 0
        numOfMeas = 24

        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::1::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
        status = niSwitch_Scan(SWITCHinstr, "com0->ch0:23;",
                               NISWITCH_VAL_DMM_INITIATED)


        REM CONFIGURE THE DMM TO TAKE 24 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_AdvancedAcquisition (DMMinstr,
                                            NIDMM_VAL_60_HERTZ,
                                            NIDMM_VAL_DC_VOLTS,
                                            dmmRange,
                                            NIDMM_VAL_TTL0,
                                            NIDMM_VAL_TTL1,
                                            triggerDelay,
                                            handInit,
                                            numOfMeas,
                                            measurements)


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

End Sub
```

B

Scanning Multiple Devices

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

Multiple device scanning is handled differently on different switch devices. Please refer to the section appropriate to the switch device you are using.

Multiple Device Scanning Using PXI Switches

For example, if you wire together the analog bus connections of multiple NI 2501 switch devices, you can have *n* times 24 channels, where *n* is the number of switch devices wired together. However,in the software, these devices are still unique switch devices with their own unique addresses. To scan in these situations, you need to:

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

Since the analog bus of multiple switch devices is wired together, setting up timing information is very critical. When scanning two devices, you need to make sure that device 1 disconnects from the analog bus before device 2 connects to the analog bus. Because of this constraint `<sa none>` must be used in the scan list. `<sa none>` allows you to delay device 2 until device 1 has disconnected from the analog bus. The usage of `<sa none>` is explained in the example below.

The NI 2501 is a 24-channel FET multiplexer that supports the analog bus connections through the PXI terminal block or terminal-block wiring. In this example, assume you have three NI 2501 devices wired together to create a 72-channel FET multiplexer. If you are scanning in Break Before Make and wanted to do a simple scan of all three devices in order, the scan lists for each device would be as follows:

```
Device #1: <sa none> ab0->ch0:23; <repeat 24>; <repeat 24>;
Device #2: <repeat 24>; <sa none> ab0->ch0:23; <repeat 24>;
Device #3: <repeat 24>; <repeat 24>; <sa none> ab0->ch0:23;
```

The number 24 in the `repeat` command is the exact number of channels scanned by the other devices. The previous example used two `repeat` command words to make this clear. However, you can combin 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, *Manually Controlling Switches*, to see how this is done.

To make sure the scan starts properly, call either `niSwitch_Scan` (niSwitch Scan.vi)or `niSwitch_Initiate Scan` (niSwitch Initiate Scan.vi), beginning with the switch devices that do not have any switch activity in the first element of the scan list—that is, device 3, then device 2, and then device 1. This way, the final call causes the switch to close, and subsequently causes the trigger that starts off the handshake.

Switch Timing

`<sa none>` can be used to add delay in the scan and is equivalent to wait for delay. In the above scan list, device 2 will wait for a programmable delay before connecting to the analog bus. The programmable delay for a device should be set to the debounce time of the previous device in the scan. This delay ensures that device 1 has disconnected from the analog bus. A simpler, but not very efficient way, is to change the delays to the worst case of all the switch devices. Device 1 scan list has `<sa none>` associated with it because if you were the doing a continuous scan, device 1 will have to wait until device 3 disconnects from the analog bus. The relevant attribute for setting the delay is Settling Time.

Note In the above example, the analog bus was the common channel between the modules involved in the scan. Depending on the application, the common channel between the devices could change. For example, you could connect a row of two matrix switch devices together where the row becomes the common channel. If you are scanning two devices (device 1 and device 2) make sure that the first channel device 1 disconnects from is the common channel.

The Trigger Configuration section explains how trigger routing is done while scanning multiple devices.

Warning A multiple device scanning system with incorrect delays can result in damage to the system or personal injury.

Multiple Device Scanning (Chain Triggering)

In this scanning mode, a switch device being scanned sends a trigger signal to the following switch device after it finishes executing its scan list. Since the analog bus of multiple switch devices will be wired together, setting up timing is very critical. While scanning two devices you need to make sure that device 1 disconnects from the analog bus before device 2 connects to the analog bus. The ability of switch devices to send triggers to particular trigger lines on the backplane is very useful in setting up the timing of the scan. Use `<sa ttlx>` to send a trigger and `<wft ttlx>` to receive a trigger, where *x* is one of the trigger lines on the backplane.

Note Currently only SCXI-1129 has the ability to send triggers on particular trigger lines.

Consider two SCXI-1129 devices in 4 x 64 mode. SCXI-1129 can be configured in various matrix configurations. Closing the analog bus relays creates a 4 x 128 matrix. If you are scanning in Break Before Make and want to do a simple scan of the two devices, the scan lists for each device could be as follows:

```
Device #1: ab0->r0 & r0->c0 & r0->c1;  <sa ttl1> <wft ttl2>
Device #2: <wft ttl1> ab0->r0 & r0->c0 & r0->c1;<sa ttl2>
```

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

In the above scan list, device 1 sends a trigger on TTL1 (`<sa ttl1>`) after disconnecting from the analog bus. Device 2 waits for this trigger (`<wft ttl1>` before it starts its scan. By having device 1 trigger device 2, you can be sure that device 1 has disconnected from the analog bus before device 2 connects to the analog bus.

`<sa ttl1>` in device 2 scan list and `<wft ttl1>` in device 1 scan list enables continuous scanning.

Multiple Device Scanning using SCXI - 1127/1128

Scanning of multiple devices using SCXI-1127 and SCXI-1128 is explained in Chapter 6, *Using NI-SWITCH with SCXI*.

Trigger Configuration

The final consideration when performing multiple device scanning is the trigger mapping. When a scan involves only a single switch device and a measurement device, the handshake triggers are simply point-to-point. However, when working with multiple switch devices, you must ensure that all boards can participate in the scan and can access the triggers. If the measurement and switch devices 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 device.

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

The Trigger Mode attribute tells the device whether or not it is operating in single, master, or slave mode. Refer to the *NI-SWITCH C Reference 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 device in a single-device scanning system and, therefore, can use the configurations as described in Chapter 5, *Scanning*. Master and slave modes are meant for device scanning.

The master switch device is the device to which the external triggers are wired.

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

- Master Slave Trigger Bus
- Master Slave Scan Advanced Bus

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

Figure B-1 illustrates this system.

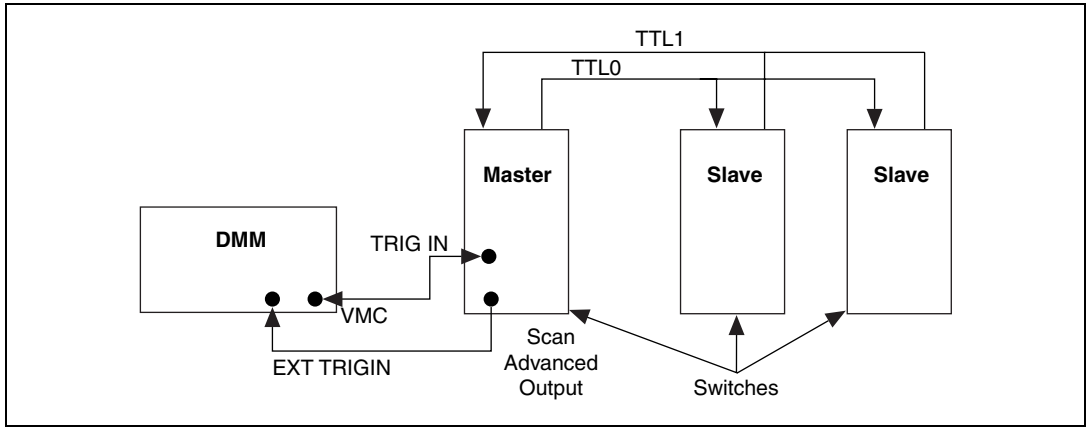

Figure B-1. External Trigger System

In this case, the attributes for both the master and slave devices would be set as shown in Table B-1.

Table B-1. Master and Slave Device Attribute Settings for External Trigger System

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

You can set the Trigger Input and Scan Advanced Output attributes for the slave devices to anything since the driver ignores them.

To continue the example described earlier, 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.

Figure B-2 illustrates this example.

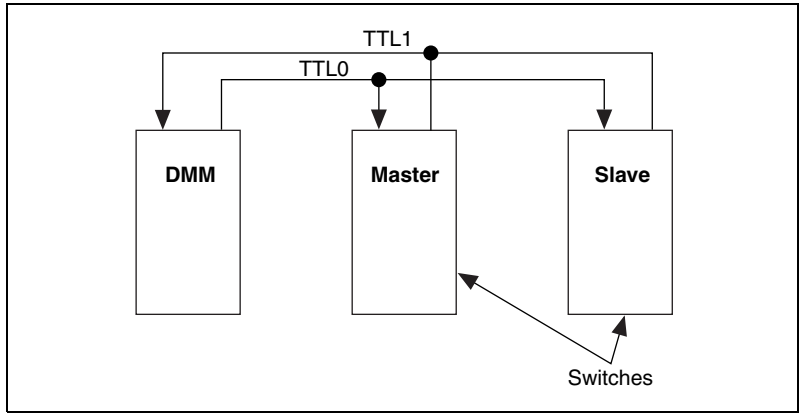

Figure B-2. Internal Trigger System

For this example, the DMM should be configured as shown in Table B-2.

Table B-2. DMM Configuration for Internal Trigger System

Trigger	Value
DMM Voltmeter Complete Trigger	TTL0
DMM Trigger Input	TTL1

The master and slave devices would then be configured as shown in Table B-3.

Table B-3. Master and Slave Device Attribute Settings for Internal Trigger System

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

Common Names Table

The following table lists the common names for both the LabVIEW VIs and C function calls that are used in this manual.

Table C-1. Common Names

Common Name	LabVIEW VI Name	C Function Call
Abort Scan	niSwitch Abort Scan	niSwitch_AbortScan
Close	niSwitch Close	niSwitch_close
Configure Scan List	niSwitch Configure Scan List	niSwitch_ConfigureScanList
Configure Scan Trigger	niSwitch Configure Scan Trigger	niSwitch_ConfigureScanTrigger
Connect	niSwitch Connect Channels	niSwitch_Connect
Disconnect	niSwitch Disconnect Channels	niSwitch_Disconnect
Disconnect All	niSwitch Disconnect All Channels	niSwitch_DisconnectAll
Error Message	niSwitch Error Message	niSwitch_ErrorMessage
Error Query	niSwitch Error Query	niSwitch_error_query
Initialize	niSwitch Initialize	niSwitch_init
Initialize With Options	niSwitch Initialize With Options	niSwitch_InitWithOptions
Initiate Scan	niSwitch Initiate Scan	niSwitch_InitiateScan
Is Debounced	niSwitch Switch Is Debounced?	niSwitch_IsDebounced
Is Scanning	niSwitch Switch Is Scanning?	niSwitch_IsScanning
Reset	niSwitch Reset	niSwitch_reset

Table C-1. Common Names (Continued)

Common Name	LabVIEW VI Name	C Function Call
`Revision Query`	niSwitch Revision Query	niSwitch_revision_query
`Scan`	niSwitch Scan	niSwitch_Scan
`Self Test`	niSwitch Self Test	niSwitch_self_test
`Send Software Trigger`	niSwitch Send Software Trigger	niSwitch_SendSoftwareTrigger
`Set Continuous Scan`	niSwitch Set Continuous Scan	niSwitch_SetContinuousScan
`Single Switch`	niSwitch Control A Single Switch	niSwitch_SingleSwitchControl
`Wait for Debounce`	niSwitch Wait for Debounce	niSwitch_WaitForDebounce
`Wait for Scan Complete`	niSwitch Wait For Scan To Complete	niSwitch_WaitForScanToComplete

D

Technical Support Resources

Web Support

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

NI Developer Zone

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

Customer Education

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

System Integration

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

Worldwide Support

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

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

Glossary

Prefix	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 device

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
Interchangeable Virtual Instruments	an advanced architecture for instrument drivers that includes features such as simulation and state caching
ISA	Industry Standard Architecture
IVI	*See* Interchangeable Virtual Instruments.

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 device	the first switch device in a multi-device scan. It is responsible for passing the triggers from the instrument to and from the slave switch device.
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 devices
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 device when scanning. The trigger occurs after the switch device 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 devices 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 device	the switch devices other than the master switch device in a multi-device 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 computer, 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

Symbols

A

B

C

D

E

G

H

I

L

M

N

O

P

R

S

T

V

W