COMPREHENSIVE SERVICES

We offer competitive repair and calibration services, as well as easily accessible documentation and free downloadable resources.

SELL YOUR SURPLUS

We buy new, used, decommissioned, and surplus parts from every NI series. We work out the best solution to suit your individual needs.

Sell For Cash Get Credit Receive a Trade-In Deal

OBSOLETE NI HARDWARE IN STOCK & READY TO SHIP

We stock **New, New Surplus, Refurbished,** and **Reconditioned** NI Hardware.

APEX WAVES

Bridging the gap *between the manufacturer and your legacy test system.*

1-800-915-6216

www.apexwaves.com

sales@apexwaves.com

Request a Quote CLICK HERE *BNC-2810*

Computer-Based Instruments

Serial Data Analyzer Software Reference Manual

Serial Data Analyzer Instrument Driver

December 1997 Edition
Part Number 321753A-01

Internet Support
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com
Telecom Group homepage: http://www.natinst.com/telecom

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

Fax-on-Demand Support
512 418 1111

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

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

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

Important Information

Warranty

The PCI/PXI-6810 Serial Data Analyzer is warranted against defects in materials and workmanship for a period of one year from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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 manual 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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual ix
Conventions Used in This Manual x
National Instruments Documentation xi
Related Documentation xi
Customer Communication xii

Chapter 1
Introduction

About the SDA Instrument Driver Software 1-1
Application Development Environments 1-1
Status Codes 1-2
Variable Data Types 1-2
Primary Types and Arrays 1-2
Programming Language Considerations 1-3
LabVIEW 1-4
LabWindows/CVI 1-4
Code Examples 1-7
Architecture 1-7
What You Need to Get Started 1-8
Software Programming Choices 1-8
National Instruments Application Software 1-8
Driver Software 1-9

Chapter 2
Using the SDA Instrument Driver

SDA Instrument Driver Interface 2-1
Fundamentals of Building Windows Applications with the SDA Instrument Driver 2-1
Creating Your Application 2-2
Building LabVIEW Applications 2-3

Software Overview 2-4
Board Functions 2-4
Life-Cycle Functions 2-4
Configuration Functions 2-5
Utility Functions 2-5
Channel Functions 2-6
Programming with the SDA Instrument Driver 2-8
C Implementation 2-9
G Implementation 2-12

Chapter 3
Functions

Board Functions 3-1
Life-Cycle Functions 3-1
niSda_init 3-2
niSda_close 3-5
Configuration Functions 3-7
niSda_SetAttribute 3-8
niSda_GetAttribute 3-11
niSda_SaveSetup 3-14
niSda_RecallSetup 3-16
Utility Functions 3-18
niSda_reset 3-19
niSda_self_test 3-21
niSda_error_query 3-23
niSda_error_message 3-25
niSda_revision_query 3-27
Channel Functions 3-29
niSda_ChannelReset 3-30
niSda_ChannelConfigureGP 3-32
niSda_ChannelConfigureRS232 3-34
niSda_ChannelConfigureRS485 3-37
niSda_ChannelSetAttribute 3-40
niSda_ChannelGetAttribute 3-43
niSda_ChannelEnableDataTransceiver 3-47
niSda_ChannelDisableDataTransceiver 3-49
niSda_ChannelEnableClockTransceiver 3-51
niSda_ChannelDisableClockTransceiver 3-53
niSda_ChannelReceive 3-55
niSda_ChannelReceiveFile 3-58
niSda_ChannelCheckReception 3-61
niSda_ChannelTransmit 3-64

niSda_ChannelTransmitPattern 3-67
niSda_ChannelTransmitFile 3-70
niSda_ChannelCheckTransmission 3-73
niSda_ChannelInsertError 3-75
niSda_ChannelAbort 3-77
niSda_ChannelTrigger 3-79

Appendix A
Attributes

Appendix B
Value Attributes

Appendix C
Status Codes

Appendix D
Customer Communication

Glossary

Index

Figures

Figure 1-1. LabVIEW VI Library 1-4
Figure 1-2. SDA Instrument Driver Architecture 1-7

Figure 2-1. LabVIEW Menu 2-3
Figure 2-2. Example Application Setup 2-8
Figure 2-3. Typical Use of Instrument Driver 2-9
Figure 2-4. Initialization and Board Configuration 2-12
Figure 2-5. Channel Configuration 2-13
Figure 2-6. Channel Transmission and Reception Operation 2-14
Figure 2-7. Closing 2-14

Tables

Table 1-1. Compatible Data Types 1-2
Table 1-2. LabWindows/CVI Function Tree for SDA 1-5

Table 2-1. Import Libraries 2-2

About This Manual

The *Serial Data Analyzer Software Reference Manual* is for users of the Serial Data Analyzer (SDA) instrument driver. The SDA instrument driver gives you programmatic control of the PCI/PXI-6810. This allows you to control individual serial channels, transceivers, and triggers within a given board and within a system based on these boards.

Organization of This Manual

The *Serial Data Analyzer Software Reference Manual* is organized as follows:

- Chapter 1, *Introduction*, describes the SDA instrument driver, lists the application development environment compatible with it, and contains important information about how to apply the functions described in this manual to your programming language and environment.
- Chapter 2, *Using the SDA Instrument Driver*, describes the basic operation of the SDA instrument driver including the functional interface and the different ADEs, and provides a detailed sample application.
- Chapter 3, *Functions*, describes the application programming interface (API) to the SDA instrument driver.
- Appendix A, *Attributes*, describes the attributes used by the SDA instrument driver.
- Appendix B, *Value Attributes*, describes the possible values for each attribute described in Appendix A, *Attributes*.
- Appendix C, *Status Codes*, describes the status codes returned by the SDA.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products or manuals.

- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. For Example, the sequence **File»Page Setup»Options» Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box.
::	This symbol separates addressable units within a virtual instrument software architecture (VISA) resource address.
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
bold	Bold text denotes the names of menus, menu items, parameters, dialog box, dialog box buttons or options, icons, windows, Windows 95 tabs, or LEDs.
bold italic	Bold italic text denotes a note, caution, or warning.
SDA	SDA refers to the serial data analyzer.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value, as in Windows 95.
`monospace`	Text in this font denotes text or characters that should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and for statements and comments taken from programs.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.

National Instruments Documentation

The *Serial Data Analyzer Software Reference Manual* is one piece of the documentation set for your SDA. You could have any of several types of manuals, depending on the hardware and software in your system. Use the different types of manuals you have as follow:

- Your hardware user manuals—These manuals have detailed information about the hardware that plugs into or is connected to your computer. This hardware may include other computer board instruments, DAQ hardware, or instrument control hardware from National Instruments. Use these manuals for hardware installation and configuration instructions, specification information about your hardware, and application hints.
- Software documentation—You may have both application software and SDA software documentation. National Instruments application software includes LabVIEW and LabWindows/CVI. After you set up your hardware system, use either the application software (LabVIEW or LabWindows/CVI) documentation, or the SDA documentation to help you write your application. If you have a large and complicated system, it is worthwhile to look through the software documentation before you configure your hardware.
- Accessory installation guides or manuals—If you are using the BNC-2810 connector block, see *Installing Your BNC-2810 SDA Connector Block*. It describes how to physically connect the relevant pieces of the system. Consult this guide when you are making your connections.

Related Documentation

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

- Technical reference manual for your computer or your PXI chassis
- National Instruments Application Note 105, *Connector Options for the 6810 Serial Data Analyzer*
- National Instruments *PXI Specification*, rev. 1.0
- *PCI Specification*, ver. 2.1
- *PICMG CompactPCI 2.0 R2.1*

- The Setup and Test guides describe how to install and configure your PCI/PXI-6810 hardware and software, and describe the connectors.
- VXI *plug&play* System Alliance documents and specifications
- LabVIEW documentation set
- LabWindows/CVI documentation set
- *Microsoft Visual C/C++ User Guide to Programming*
- NI-DAQ documentation set

Customer Communication

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

Chapter 1

Introduction

This chapter describes the SDA instrument driver, lists the application development environment compatible with it, and contains important information about how to apply the functions described in this manual to your programming language and environment.

About the SDA Instrument Driver Software

Thank you for buying a National Instruments SDA device for the PCI/PXI bus. Your purchase includes the SDA instrument driver—a set of functions that control the National Instruments plug-in SDA device for analysis of digital serial data.

The SDA instrument driver follows the VXI plug and play instrument driver model and provides low-level and application-level functions.

Application Development Environments

This release of the SDA instrument driver supports the following application development environments (ADEs) for Windows 95 and Windows NT:

- LabVIEW version 4.*x* or later
- LabWindows/CVI version 4.*x* or later
- Borland C/C++ version 4.5.*x* or 5.*x* or later
- Microsoft Visual C/C++ version 4.*x* or 5.*x* or later

☞ **Note:** ***Although the SDA instrument driver has been tested and found to work with these ADEs, other ADEs or higher versions of the ADEs listed above may also work.***

The SDA instrument driver software comes to you on a CD-ROM. Always run the SDA instrument driver installation utility to extract the files you want. For a brief description of the directories produced by the install programs and the names and purposes of the files, consult the `readme.txt` file.

Status Codes

Every SDA instrument driver function is of the following form:

rval = `Function_Name` (parameter 1, parameter 2, ... parameter *n*)

where *n* > 0. Each function returns a status code (**rval**) that indicates the success or failure of the function, as discussed in Appendix B, *Value Attributes*. All error codes are mapped as negative values, while success values are mapped as zero or positive.

Variable Data Types

The SDA instrument driver application programming interface (API) is identical in Windows 95 and Windows NT. LabWindows/CVI uses the same data types as Windows 95 and Windows NT. The following sections describe the notation used in those parameter tables and throughout the manual for variable data types.

Primary Types and Arrays

Table 1-1 shows the primary type names and their ranges.

Table 1-1. Compatible Data Types

Type Name	Direction	Definition
`ViBoolean`	IN	Boolean value
`ViPBoolean`	OUT	Pointer to a `ViBoolean` value
`ViBoolean[]`	IN/OUT	Pointer to an array of `ViBoolean` values
`ViInt16`	IN	Signed 16-bit integer
`ViPInt16`	OUT	Pointer to a `ViInt16` value
`ViInt16[]`	IN/OUT	Pointer to an array of `ViInt16` values
`ViInt32`	IN	Signed 32-bit integer
`ViPInt32`	OUT	Pointer to a `ViInt32` value
`ViInt32[]`	IN/OUT	Pointer to an array of `ViInt32` values
`ViReal64`	IN	64-bit floating-point number
`ViPReal64`	OUT	Pointer to a `ViReal64` value
`ViReal64[]`	IN/OUT	Pointer to an array of `ViReal64` values

Table 1-1. Compatible Data Types (Continued)

Type Name	Direction	Definition

`ViString`
`ViPString`
`ViChar[]`
`ViRsrc`
`ViPRsrc`
	IN
	OUT
	IN/OUT
	IN
	OUT
		Pointer to a C string
		Pointer to a C string
		Pointer to a C string
		A VISA resource descriptor (`ViString`)
		Pointer to a `ViRsrc` value (`ViString`)

`ViSession`
`ViPSession`
`ViSession[]`
	IN
	OUT
	IN/OUT
		A VISA session handle
		Pointer to a `ViSession`
		Pointer to an array of `ViSession` values

`ViStatus`	IN	A VISA return status type

☞ **Note:** ***The types specified as*** `Vi<Type>[]` ***in Table 1-1 are symbolic descriptions for single dimension arrays of the specified type and are not the correct syntactic representation.***

Programming Language Considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the SDA instrument driver API.

☞ **Note:** ***Be sure to include the SDA instrument driver function prototypes by including the appropriate SDA instrument driver header file in your source code.***

LabVIEW

Inside the LabVIEW environment, the SDA instrument driver VIs appear in the instrument driver menu. Each VI encapsulates an instrument driver function. For information on how to use LabVIEW VIs with your SDA system, refer to Chapter 2, *Using the SDA Instrument Driver*. Figure 1-1 shows the SDA VIs available in the LabVIEW library.

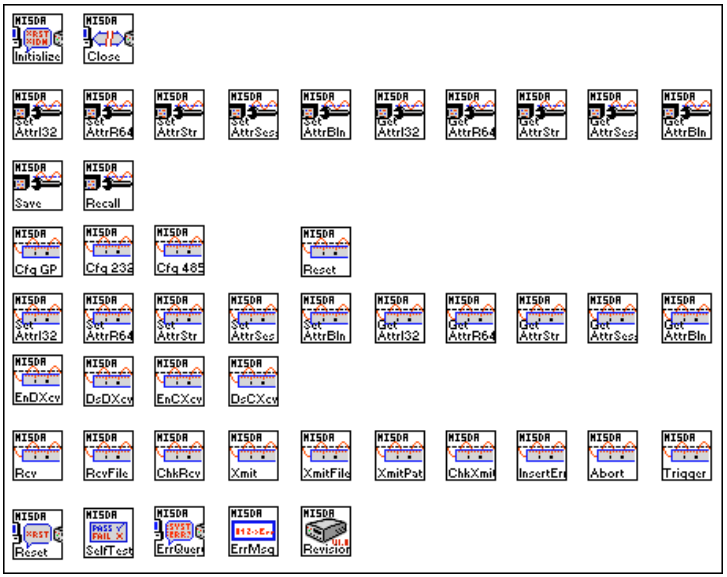


Figure 1-1. LabVIEW VI Library

LabWindows/CVI

Inside the LabWindows/CVI environment, the SDA instrument driver functions appear in the **Instrument Driver** menu. Each function panel represents an SDA instrument driver function, which is displayed at the bottom of the panel.

For information on how to use LabWindows/CVI with your SDA system, refer to Chapter 2, *Using the SDA Instrument Driver*.

Table 1-2 shows how the LabWindows/CVI function panel tree is organized, and the SDA instrument driver function name that corresponds to each function panel.

Table 1-2. LabWindows/CVI Function Tree for SDA

Description	Function
Board Functions	
Life Cycle Functions	
Initialize	niSda_init
Close	niSda_close
Configuration	
Set Attribute ViBoolean	niSda_SetAttributeViBoolean
Set Attribute ViInt32	niSda_SetAttributeViInt32
Set Attribute ViReal64	niSda_SetAttributeViReal64
Set Attribute ViSession	niSda_SetAttributeViSession
Set Attribute ViString	niSda_SetAttributeViString
Get Attribute ViBoolean	niSda_GetAttributeViBoolean
Get Attribute ViInt32	niSda_GetAttributeViInt32
Get Attribute ViReal64	niSda_GetAttributeViReal64
Get Attribute ViSession	niSda_GetAttributeViSession
Get Attribute ViString	niSda_GetAttributeViString
Save Setup	niSda_SaveSetup
Recall Setup	niSda_RecallSetup
Utility	
Reset	niSda_reset
Self-Test	niSda_self_test
Error-Query	niSda_error_query
Error Message	niSda_error_message
Revision Query	niSda_revision_query
Channel Functions	
Configure	
Reset	niSda_ChannelReset
Configure General Purpose	niSda_ChannelConfigureGP

Table 1-2. LabWindows/CVI Function Tree for SDA (Continued)

Description	Function
Configure RS-232	niSda_ChannelConfigureRS232
Configure RS-485	niSda_ChannelConfigureRS485
Attributes	
Set Attribute ViBoolean	niSda_ChannelSetAttributeViBoolean
Set Attribute ViInt32	niSda_ChannelSetAttributeViInt32
Set Attribute ViReal64	niSda_ChannelSetAttributeViReal64
Set Attribute ViSession	niSda_ChannelSetAttributeViSession
Set Attribute ViString	niSda_ChannelSetAttributeViString
Get Attribute ViBoolean	niSda_ChannelGetAttributeViBoolean
Get Attribute ViInt32	niSda_ChannelGetAttributeViInt32
Get Attribute ViReal64	niSda_ChannelGetAttributeViReal64
Get Attribute ViSession	niSda_ChannelGetAttributeViSession
Get Attribute ViString	niSda_ChannelGetAttributeViString
Transceiver	
Enable Data Transceiver	niSda_ChannelEnableDataTransceiver
Disable Data Transceiver	niSda_ChannelDisableDataTranceiver
Enable Clock Transceiver	niSda_ChannelEnableClockTranceiver
Disable Clock Transceiver	niSda_ChannelDisableClockTranceiver
Transmission	
Receive	niSda_ChannelReceive
Receive File	niSda_ChannelReceiveFile
Check Reception	niSda_ChannelCheckReception
Transmit	niSda_ChannelTransmit
Transmit Pattern	niSda_ChannelTransmitPattern
Transmit File	niSda_ChannelTransmitFile
Check Transmission	niSda_ChannelCheckTransmission
Insert Error	niSda_ChannelInsertError
Abort	niSda_ChannelAbort
Trigger	
Trigger	niSda_ChannelTrigger

Code Examples

You can find code examples in the same directory in which you installed the SDA instrument driver software. You can find source code common to all environments in the `Examples` subfolder.

Architecture

A block diagram of the SDA instrument driver architecture shown in Figure 1-2 illustrates the low- and mid-level architecture for SDA devices.

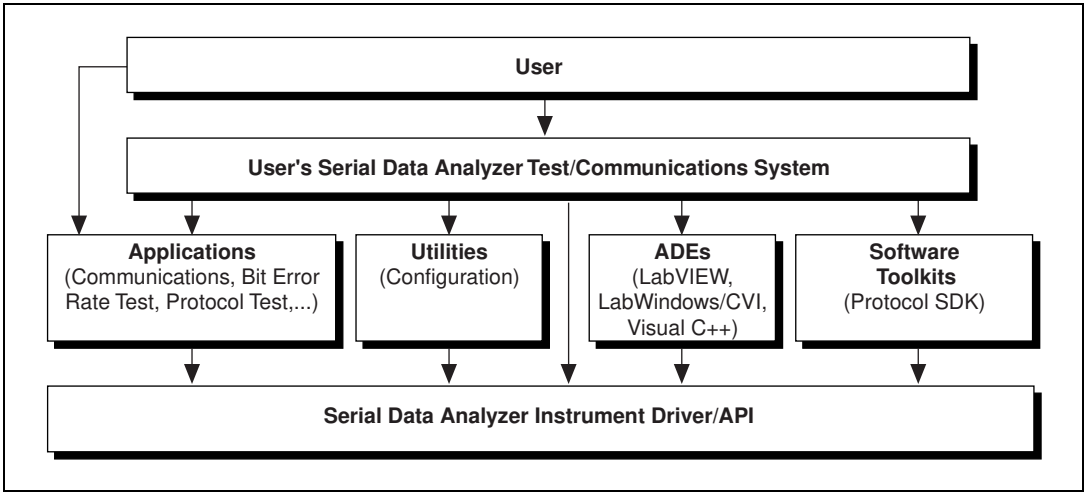


Figure 1-2. SDA Instrument Driver Architecture

The architecture uses a *hardware abstraction layer,* which separates software API capabilities, such as general acquisition and control functions, from hardware-specific information. This layer lets you use new SDA hardware without having to recompile your applications.

What You Need to Get Started

To set up and use your 6810 device, you need the following:

- ❑ PCI-6810 or PXI-6810 device
- ❑ *Getting Started with Your PCI/PXI-6810 Serial Data Analyzer* manual
- ❑ SDA instrument driver for Windows 95/NT
- ❑ Optional software packages and documentation:
  - LabVIEW
  - LabWindows/CVI
- ❑ Your Pentium-based PCI computer running Windows 95 or Windows NT, version 4.0 or later
- ❑ PXI chassis (with PXI-6810 only)

Software Programming Choices

You have several options to choose from when programming your National Instruments SDA hardware. You can use National Instruments application software such as LabVIEW and LabWindows/CVI, National Instruments SDA instrument driver software. You can use other third-party ADEs such as Borland C/C++ and Microsoft Visual C/C++ to interface the SDA software to standard Windows DLLs.

National Instruments Application Software

LabVIEW features interactive graphics, a state-of-the-art user interface, and a powerful graphical programming language. The LabVIEW SDA VI Library, a series of virtual instruments (VIs) for using LabVIEW with National Instruments SDA hardware, is included with your software kit.

LabWindows/CVI features interactive graphics, a state-of-the-art user interface, and uses the ANSI standard C programming language. The LabWindows/CVI SDA Library, a series of functions for using LabWindows/CVI with National Instruments SDA hardware, is included with your software kit.

You can also use ComponentWorks, a state-of-the-art user interface component, with many third-party ADEs.

Driver Software

The Serial Data Analyzer instrument driver software is included at no charge with the 6810. The SDA driver software has an extensive library of functions that you can call from your application programming environment. These functions include transmitting and receiving serial data streams. The SDA instrument driver software performs all functions required for acquiring and saving serial data analysis.

The SDA instrument driver software has functions for maximum flexibility and performance. Examples of high-level functions include the functions to acquire serial data analysis in single-shot or continuous mode to a file. An example of a low-level function is configuring a serial data analysis sequence, since it requires advanced understanding of the 6810 device and serial data analysis.

The SDA instrument driver software also internally resolves many of the complex issues between the computer and the 6810 device, such as programming interrupts and DMA controllers. The SDA instrument driver software is the interface path between LabVIEW, LabWindows/CVI, or a conventional programming environment and the 6810 device.

Any platform that supports the SDA instrument driver integrates with NI-DAQ and a variety of National Instruments DAQ devices, so you can integrate your 6810 device and 6810 instrument driver development can integrate with National Instruments DAQ products.

Using the SDA Instrument Driver

Chapter 2

This chapter describes the basic operation of the SDA instrument driver including the functional interface and the different ADEs, and provides a detailed sample application.

SDA Instrument Driver Interface

The SDA instrument driver follows the VXI *plug&play* System Alliance instrument driver model, which means that it uses a standard format for the arrangement of different functions. This model breaks the functions into board functions and channel functions. The board functions address the entire instrument and are mainly concerned with configuration. The channel functions address the individual channels within the instrument and allow you to transmit and receive data on specific transceivers.

Fundamentals of Building Windows Applications with the SDA Instrument Driver

The SDA instrument driver for the Windows 95/NT function library is a dynamic link library (DLL), which means that instrument driver routines are not linked into the executable files of applications. Only the information about the instrument driver routines in the instrument driver import libraries is stored in the executable files.

Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Depending on the development tools you are using, you may give the DLL routines information through import libraries or through function declarations. Your SDA instrument driver software kit contains function prototypes for all routines.

Creating Your Application

This section outlines the process for developing SDA instrument driver applications using C for Windows 95 and Windows NT. Detailed instructions on creating project and source files are not included. For information on creating and managing project files, consult the documentation included with your particular development environment.

When programming, use the following guidelines:

- All C source files that use SDA instrument driver functions must include the `NISDA.H` header file. Add this file to the top of your source files.
- You must add the `NISDA.LIB` import library to your project. Some environments allow you to add import libraries simply by inserting them into your list of project files. Other environments allow you to specify import libraries under the linker settings portion of the project file.
- When compiling, you will need to indicate where the compiler can find the NI-SDA header files and shared libraries. Most of the files you need for development are located under the NI-SDA target installation directory. If you choose the default directory during installation, the target installation directory is `C:\NISDA`. The include files are located under the `include` subdirectory. The import libraries are located under the `lib\<environment>` subdirectory for the platforms shown in Table 2-1.

Table 2-1. Import Libraries

Development Environment	Directory
Microsoft Visual C++	`lib\msc`
Borland C++	`lib\bc`

The soft front panel is located in the `nisda` subdirectory.

Building LabVIEW Applications

All VIs dedicated to the SDA PCI/PXI-6810 board are in the `nisda.llb` library. When you choose the **Instrument Driver»NI SDA** menu, you will see a palette similar to the one shown in Figure 2-1.

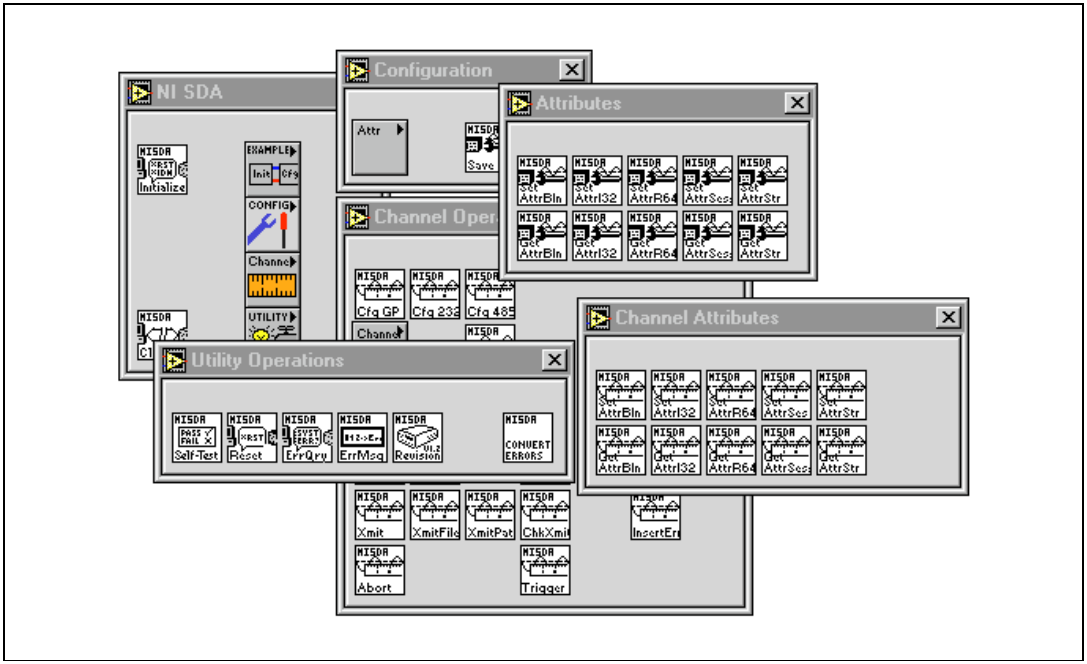


Figure 2-1. LabVIEW Menu

All SDA instrument driver functions are called using the LabVIEW Call Library Function. With good knowledge of instrument drivers, you can easily create user specific VIs that map directly to the functionality you require.

You can use the SDA instrument driver from a number of ADEs such as LabVIEW, LabWindows/CVI, and Visual C/C++. The API is designed for easy use, yet provides you with the flexibility of accessing every feature of the instrument. This chapter includes a simple application that implements the API in C (using LabWindows/CVI and Visual C/C++) and G (using LabVIEW). See *Application Development Environments*, in Chapter 1, *Introduction*, for a list of the versions of currently supported ADEs.

You can find source code to all of the implementations in the installation directory. Please refer to the `readme.txt` file located in your target installation directory for the latest details on SDA instrument driver sample programs. These programs are installed in the `Examples` subdirectory under the target installation folder if you elected to install the sample files.

Software Overview

This section describes the classes of the SDA instrument driver functions according to the following classes:

- Board functions
  - Life-cycle
  - Configuration
  - Utility
- Channel functions
  - Configuration
  - Transceiver
  - Transmission
  - Trigger

Board Functions

Use the board functions for setup and configuration.

Life-Cycle Functions

`niSda_init`	Establishes a communication session to the instrument.
`niSda_close`	Closes the current session to the instrument.

Configuration Functions

`niSda_SetAttribute<type>`	Sets an attribute of type `ViBoolean`. Sets an attribute of type `ViInt32`. Sets an attribute of type `ViReal64`. Sets an attribute of type `ViSession`. Sets an attribute of type `ViChar[]`.
`niSda_GetAttributeVi<type>`	Gets an attribute of type `ViBoolean`. Gets an attribute of type `ViInt32`. Gets an attribute of type `ViReal64`. Gets an attribute of type `ViSession`. Gets an attribute of type `ViChar[]`.
`niSda_SaveSetup`	Saves the current instrument setup to the given `location`.
`niSda_RecallSetup`	Recalls a previously saved instrument setup from the given `location`.

Utility Functions

`niSda_reset`	Places the instrument in a default state.
`niSda_self_test`	Causes the instrument to perform a self-test and returns the result of that self-test.
`niSda_error_query`	Queries the instrument and returns instrument-specific error information.
`niSda_error_message`	Translates the error return value from the SDA instrument driver function to a user-readable string.
`niSda_revision_query`	Returns the revision of the instrument driver and the firmware revision of the instrument being used.

Channel Functions

Use the channel functions for channel setup and control.

`niSda_ChannelReset` Places the given channel in the default state.

`niSda_ChannelConfigureGP`
Sets channel attributes frequently used in applications that use the general-purpose transceivers.

`niSda_ChannelConfigureRS232`
Sets channel attributes frequently used in applications that use the RS-232 transceivers.

`niSda_ChannelConfigureRS485`
Sets channel attributes frequently used in applications that use the RS-485 transceivers.

`niSda_ChannelSetAttribute<type>`
Sets a channel attribute of type `ViBoolean`.
Sets a channel attribute of type `ViInt32`.
Sets a channel attribute of type `ViReal64`.
Sets a channel attribute of type `ViSession`.
Sets a channel attribute of type `ViChar[]`.

`niSda_ChannelGetAttribute<type>`
Gets a channel attribute of type `ViBoolean`.
Gets a channel attribute of type `ViInt32`.
Gets a channel attribute of type `ViReal64`.
Gets a channel attribute of type `ViSession`.
Gets a channel attribute of type `ViChar[]`.

`niSda_ChannelEnableDataTransceiver`
Enables the data transceiver for input, output, or input/output on a given channel depending on the mode selected.

`niSda_ChannelDisableDataTransceiver`
Disables the data transceiver for input, output, or input/output on a given channel depending on the `mode` selected.

`niSda_ChannelEnableClockTransceiver`
Enables the clock transceiver for input, output, or input/output on a given channel depending on the `mode` selected.

`niSda_ChannelDisableClockTransceiver`
Disables the clock transceiver for input, output, or input/output on a given channel depending on the `mode` selected.

`niSda_ChannelReceive` Enables the SDA to receive data to a buffer on a given channel.

`niSda_ChannelReceiveFile`
Enables the SDA to receive data to a file on a given channel

`niSda_ChannelCheckReception`
Checks the progress of the last current reception on a given channel.

`niSda_ChannelTransmit` Enables the SDA to transmit data from a buffer on a given channel.

`niSda_ChannelTransmitPattern`
Enables the SDA to transmit from a pre-configured pattern.

`niSda_ChannelTransmitFile`
Enables the SDA to transmit data from a file on a given channel.

`niSda_ChannelCheckTransmission`
Checks the progress of the current transmission on a given channel.

`niSda_ChannelInsertError`
Inserts an error of type **errorType** onto a given channel.

`niSda_ChannelAbort` Aborts the last reception or transmission on a given channel.

`niSda_ChannelTrigger` Triggers a previously armed condition on a given channel.

Programming with the SDA Instrument Driver

The example application covers some of the basic features of the SDA and serves as a typical-use case. When you are familiar with this programming style, you will find that other applications follow a structure similar to the example code.

The example application has the setup shown in Figure 2-2 with the SDA connected to a serial communications device. The voltages on the line are RS-232 compatible on one channel, and +1 V low, +6 V high, and +2.5 V input threshold on the other channel. A clock line is also connected and is controlled by the serial device.

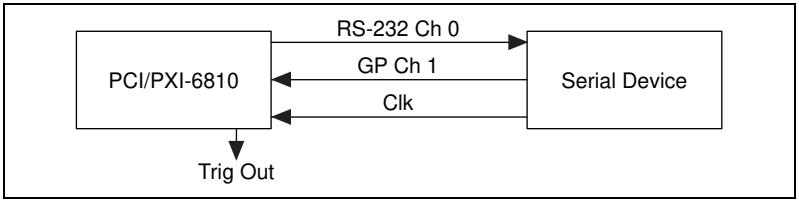


Figure 2-2. Example Application Setup

In this example, the serial device represents a device that normally receives data, processes it, and then sends some transformation of the data on a second port. In this case, it also converts the signal levels from RS-232 levels to a custom level of +1 V (low) to +6 V (high). Therefore, the PCI/PXI-6810 standard RS-232 connects to send the data and then captures it from the second port using the custom voltage levels on the general-purpose transceivers. Since the communication is synchronous, the clock connects from the serial device to the PCI/PXI-6810. In this

example, the PCI/PXI-6810 captures a particular pattern coming from the output of the serial device. This special pattern occurs only when there is a transformation problem on the serial device. The PCI/PXI-6810 also sends a trigger on the TrigOut pin to alert other monitoring equipment. Please refer to the code below to see the implementation of such a program in both C and LabVIEW. The complete source code is included in the software distribution in the `NISDA Test Example\` directory. The code follows the typical use model of the instrument driver shown in Figure 2-3.

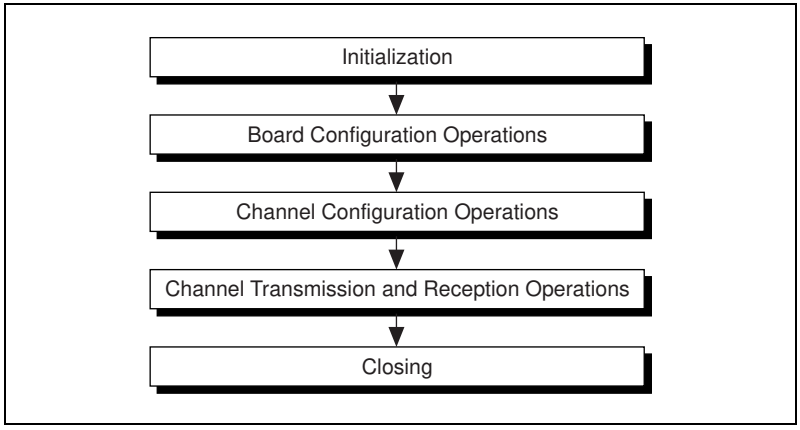


Figure 2-3. Typical Use of Instrument Driver

C Implementation

The C implementation of the example was developed using CVI 4.01. You can also use newer versions of LabWindows or other ADEs like Microsoft Visual C/C++ to compile the example code. An appropriate project file is included in the distribution.

```
#include <ansi_c.h>
#include <utility.h>
#include "NISDA.h"

static ViUInt8 dataInit[] = "\x00\x01\x02\x03\x04\x05\x06\x07";

int main (int argc, char *argv[])
{
  ViChar rsrcName[256];/* Resource name */
  ViInt16 devNum;
  ViSession dsaInstr;/* Instrument session */
  ViChar buffer[256];/* Element buffer */
  ViUInt32 numElementsActuallyReceived;/* Elements received */
```

```
ViReal64 rTime;/* Max receive time */
ViStatus status;
int testResult = 0;

/* CVI INITIALIZATION */

  if (InitCVIRTE (0, argv, 0) == 0)  /* Needed if linking in external
    compiler */
      return -1;    /* out of memory */

/* BOARD INITIALIZATION */

/* Open first SDA instrument on PCI bus 0 */
for (devNum=0; devNum<32; devNum++)
{
  sprintf(rsrcName, "PCI0::%d::INSTR", devNum);
  if (niSda_init (rsrcName, NISDA_VAL_ON, NISDA_VAL_ON, &dsaInstr) >=
    NISDA_SUCCESS)
      break;
  else
    return -1;// Exit application if failed
}

/* BOARD CONFIGURATION OPERATIONS */

/* Select firmware SOI*/
status = niSda_SetAttributeViString(dsaInstr,
         NISDA_ATTR_FIRMWARE_PROGRAM, "SOI");

/* CHANNEL CONFIGURATION OPERATIONS */

/* Setup the send channel */

status = niSda_ChannelConfigureRS232 (dsaInstr, "0",
         NISDA_VAL_XCEIVER_RS232CH0, 38400, 7, NISDA_VAL_ASRL_PAR_EVEN, 2,
         NISDA_VAL_ASRL_FLOW_NONE);

/* Setup the receive channel */
status = niSda_ChannelConfigureGP (dsaInstr, "1",
         NISDA_VAL_XCEIVER_PCH1, NISDA_VAL_TIMING_SYNC,
         NISDA_VAL_CLOCK_EXTERNAL, NISDA_VAL_XCEIVER_XV);
status = niSda_ChannelSetAttributeViReal64(dsaInstr, "1",
         NISDA_CHANNEL_ATTR_XCEIVER_THV, 2.5);

/* Setup match pattern for response and arm channel 1 */
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
         NISDA_CHANNEL_ATTR_TRIG_COND, NISDA_VAL_TRIG_COND_A);
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
         NISDA_CHANNEL_ATTR_TRIG_COND_A,
         NISDA_VAL_TRIG_COND_PATTERN_MATCH);
```

```
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
        NISDA_CHANNEL_ATTR_TRIG_PATTERN_MATCH_63_32, 0);
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
        NISDA_CHANNEL_ATTR_TRIG_PATTERN_MATCH_31_0, 0x4E90000);/* : */
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
        NISDA_CHANNEL_ATTR_TRIG_PATTERN_MASK_63_32, 0);
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
        NISDA_CHANNEL_ATTR_TRIG_PATTERN_MASK_31_0, 0x7FF0000);
status = niSda_ChannelSetAttributeViInt32(dsaInstr, "1",
        NISDA_CHANNEL_ATTR_TRIG_ROUTE, NISDA_VAL_TRIG_ROUTE_TRIGOUT_PIN);

/* CHANNEL TRANSMISSION AND RECEPTION OPERATIONS */

/* Send initial data */
status = niSda_ChannelTransmit (dsaInstr, "1", dataInit, sizeof(dataInit),
        NISDA_VAL_SENDMODE_SINGLE, NULL);

/* Initiate receive */
status = niSda_ChannelReceive(dsaInstr, "1", 256,
        NULL, NISDA_VAL_RECVMODE_DEFERRED | NISDA_VAL_RECVMODE_SINGLE,
        NULL);

/* Dial on channel 0 and wait for response on channel 1 */

status = niSda_ChannelTransmitFile (dsaInstr, "0",
"dataFile", NISDA_VAL_ENTIRE_FILE, NISDA_VAL_SENDMODE_CONTINUOUS, NULL);
rTime = Timer();
while (niSda_ChannelCheckReception(dsaInstr, "1",
256, buffer, 0, &numElementsActuallyReceived) == NISDA_SUCCESS_IO_PENDING)
{
  if (Timer() > (rTime + 45.0))
  {
    status = niSda_ChannelAbort(dsaInstr, "1", 0);
    testResult = -1;
    goto testClosing;
  }
}
```

```
  /* TEST RESULT EVALUATION */
  /* Evaluate data after trigger and set testResult here */

  /* CLOSING */
testClosing:

  /* Stop channel transmission */
  status = niSda_ChannelAbort(dsaInstr, "0", 0);

  /* Close existing instrument handle */
  niSda_close(dsaInstr);

return testResult;
}
```

G Implementation

The LabVIEW implementation of the example shown in Figure 2-4, Figure 2-5, Figure 2-6, and Figure 2-7 was developed using LabVIEW 4.1. You can also use newer versions of LabVIEW to compile your sample code.

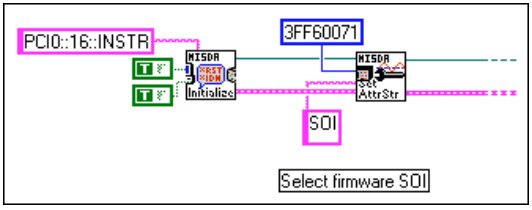


Figure 2-4. Initialization and Board Configuration

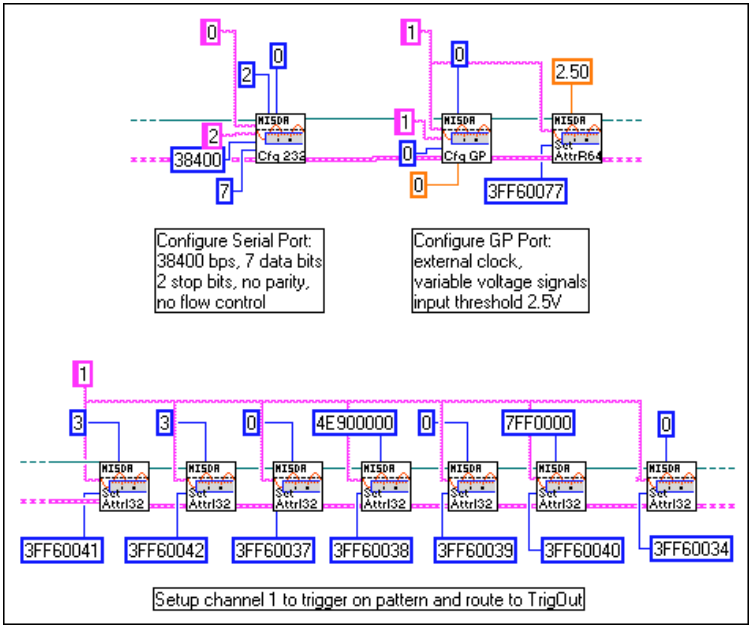


Figure 2-5. Channel Configuration

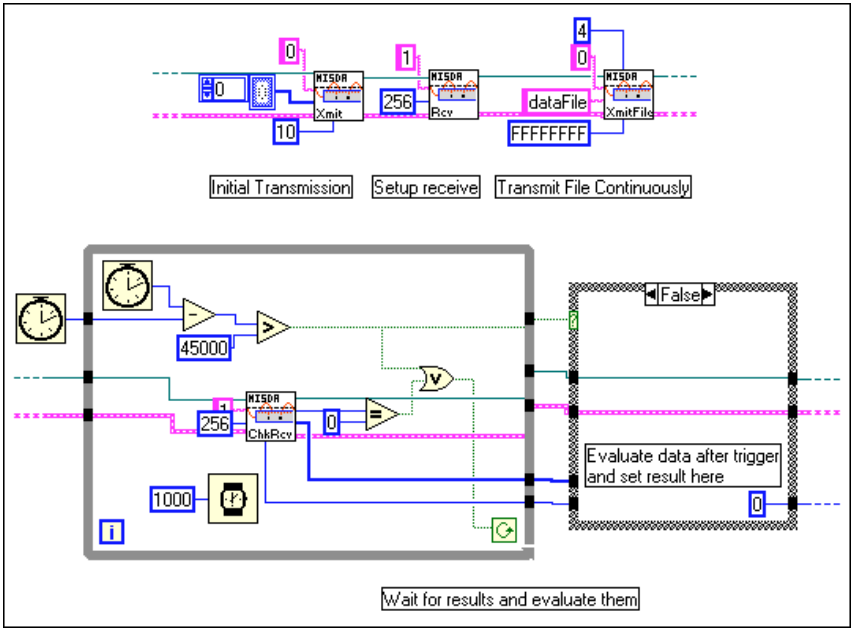


Figure 2-6. Channel Transmission and Reception Operation

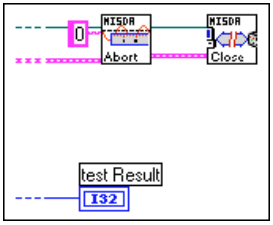


Figure 2-7. Closing

☞ **Note:** ***Look in the LabVIEW pop-up help for each set attribute and get attribute VI to see what attributes are supported and to determine their IDs.***

Chapter 3

Functions

This chapter describes the application programming interface (API) to the SDA instrument driver. This instrument driver follows the structure and paradigm of the VXI Plug&Play Systems Alliance. As such, the interface is divided into two logical sections: board-level functionality, and channel-level functionality.

Board Functions

Board functions include `niSda_init`, `niSda_SetAttributeViBoolean`, `niSda_SetAttributeViInt32`, `niSda_SetAttributeViReal64`, `niSda_SetAttributeViSession`, `niSda_SetAttributeViString`, `niSda_GetAttributeViBoolean`, `niSda_GetAttributeViInt32`, `niSda_GetAttributeViReal64`, `niSda_GetAttributeViSession`, `niSda_GetAttributeViString`, `niSda_SaveSetup`, `niSda_RecallSetup`, `niSda_reset`, `niSda_self_test`, `niSda_error_query`, `niSda_error_message`, `niSda_revision_query`, and `niSda_close`.

The functions described in this section have instrument board-wide scope. They can be further divided into life-cycle, configuration, and utility functions.

Life-Cycle Functions

The life-cycle functions (`niSda_init` and `niSda_close`) allow you to open and close a session to the instrument.

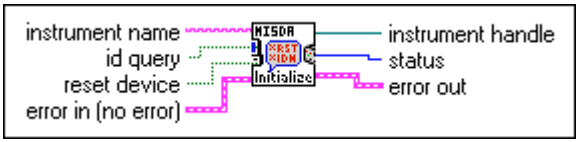
niSda_init

Format

C Language

rval = `niSda_init` **(ViString instrumentName, ViBoolean IDQuery, ViBoolean resetDevice, ViSession *instrumentHandle)**

G Language



NISDA Init.vi

Purpose

This function establishes a communication session to the instrument. It performs the following initialization actions:

- Opens a session to the specified device using the interface and address specified in the **instrumentName** parameter.
- Performs an identification query on the instrument.
- Resets the instrument to a known state.
- Returns an instrument handle that is used to differentiate between sessions of this instrument driver.

Each time you invoke this function a unique session is opened. It is possible to have more than one session open for the same resource.

niSda_init

continued

Parameters

Name	Type	Direction	Description
instrumentName	`ViString`	Input	Instrument description
idQuery	`ViBoolean`	Input	ID Query
resetDevice	`ViBoolean`	Input	Reset device
***instrumentHandle**	`ViSession`	Output	Instrument handle

Error in and error out refer to LabVIEW error cluster input and output.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_init

continued

Parameter and Return Value Discussion

instrumentName is an instrument description. The string has the following format **PCI[bus]::dev[::func]::INSTR**, where bus describes the PCI/PXI bus on which the board is located, **dev** describes the device ID, and **func** the function ID. **bus** and **func** default to 0 when omitted.

idQuery if (**VI_TRUE**), perform In-System Verification. If (**VI_FALSE**), do not perform In-System Verification.

resetDevice if (**VI_TRUE**), perform a reset operation. If (**VI_FALSE**), do not perform reset operation.

***instrumentHandle** is an instrument handle. This value is passed as the instrument handle to any other calls. It returns as VI_NULL if the SDA was not initialized successfully.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

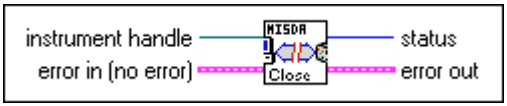
niSda_close

Format

C Language

rval = `niSda_close` **(ViSession instrumentHandle)**

G Language



NISDA Close.vi

Purpose

Closes the current session to the instrument. This implies the termination of any transmission or reception that was started within this session. Once this session is closed, the value of `vi` is invalid. All other sessions remain unaffected.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle

Error in and error out refer to LabVIEW error cluster input and output.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_close

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

Configuration Functions

The configuration functions (`niSda_SetAttributeViBoolean`, `niSda_SetAttributeViInt32`, `niSda_SetAttributeViReal64`, `niSda_SetAttributeViSession`, `niSda_SetAttributeViString`, `niSda_GetAttributeViBoolean`, `niSda_GetAttributeViInt32`, `niSda_GetAttributeViReal64`, `niSda_GetAttributeViSession`, `niSda_GetAttributeViString`, `niSda_SaveSetup`, and `niSda_RecallSetup`) allow you to set and get attributes that affect a board, and to save and recall setup to and from persistent storage.

niSda_SetAttribute

Format

C Language

rval = `niSda_SetAttributeViBoolean` **(ViSession instrumentHandle, ViAttr attributeID, ViBoolean value)**

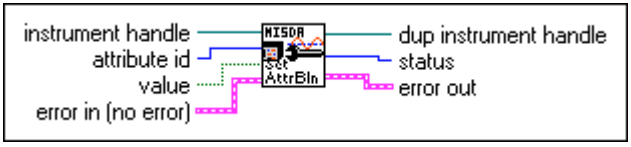
rval = `niSda_SetAttributeViInt32` **(ViSession instrumentHandle, ViAttr attributeID, ViInt32 value)**

rval = `niSda_SetAttributeViReal64` **(ViSession instrumentHandle, ViAttr attributeID, ViReal64 value)**

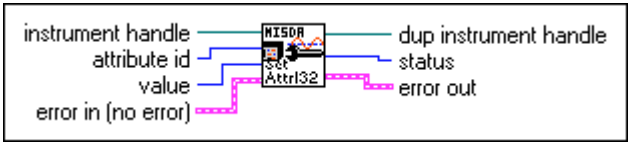
rval = `niSda_SetAttributeViSession` **(ViSession instrumentHandle, ViAttr attributeID, ViSession value)**

rval = `niSda_SetAttributeViString` **(ViSession instrumentHandle, ViAttr attributeID, ViChar value[])**

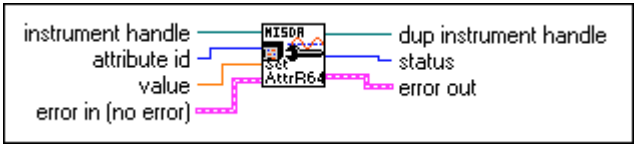
G Language



NISDA SetAttributeBoolean.vi



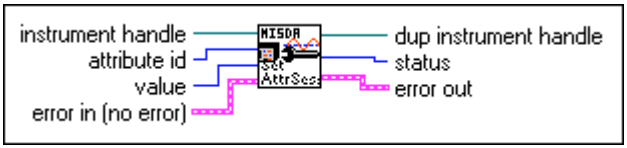
NISDA SetAttributeInt32.vi



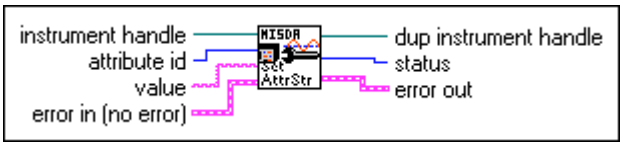
NISDA SetAttributeReal64.vi

niSda_SetAttribute

continued



NISDA SetAttributeSession.vi



NISDA SetAttributeString.vi

Purpose

These functions set an attribute of type `ViBoolean`, `ViInt32`, `ViReal64`, `ViSession`, or `ViChar[]` depending on the function used.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
attributeID	`ViAttr`	Input	Attribute for which the state is to be modified

value	`ViBoolean`
	`ViInt32`
	`ViReal64`
	`ViSession`
	`ViChar[]`
		Input	The state of the attribute

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_SetAttribute

continued

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init.

attributeID is an attribute for which the state is to be modified. For the list of attributes see Appendix A, *Attributes*.

value is the state of the attribute. The valid values depend on the value of attributeID as described in Appendix A, *Attributes*.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

niSda_GetAttribute

Format

C Language

rval = `niSda_GetAttributeViBoolean` **(ViSession instrumentHandle, ViAttr attributeID, ViBoolean *value)**

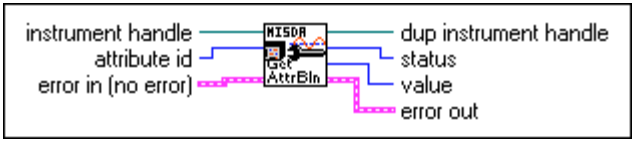
val = `niSda_GetAttributeViInt32` **(ViSession instrumentHandle, ViAttr attributeID, ViInt32 *value)**

rval = `niSda_GetAttributeViReal64` **(ViSession instrumentHandle, ViAttr attributeID, ViReal64 *value)**

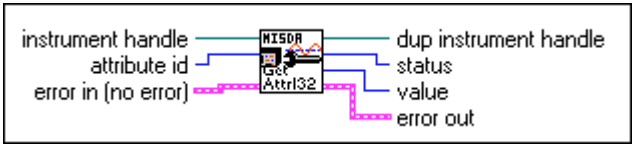
rval = `niSda_GetAttributeViSession` **(ViSession instrumentHandle, ViAttr attributeID, ViSession *value)**

rval = `niSda_GetAttributeViString` **(ViSession instrumentHandle, ViAttr attributeID, ViInt32 bufferSize, ViChar value[])**

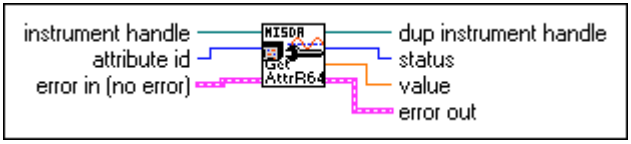
G Language



NISDA GetAttributeViBoolean.vi



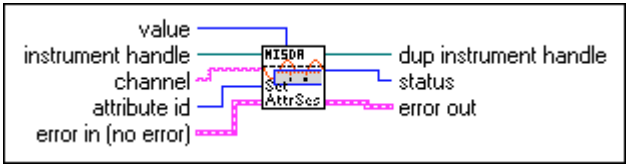
NISDA GetAttributeViInt32.vi



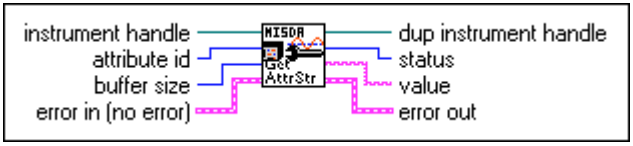
NISDA GetAttributeViReal64.vi

niSda_GetAttribute

continued



NISDA GetAttributeViSession.vi



NISDA GetAttributeViString.vi

Purpose

These functions get an attribute of type `ViBoolean`, `ViInt32`, `ViReal64`, `ViSession`, or `ViChar[]` depending on the function used.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
attributeID	`ViAttr`	Input	Attribute for which the state is to be modified

***value**	`ViBoolean`
	`ViInt32`
	`ViReal64`
	`ViSession`
	`ViChar[]`
		Input	The state of the attribute

bufferSize	`ViInt32`	Input	Maximum size of an array returned in value

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_GetAttribute

continued

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

attributeID is an attribute for which the state is to be modified. For the list of attributes see Appendix A, *Attributes*.

bufferSize is the maximum size of an array returned in value.

value is the state of the attribute. The valid values depend on the value of `attributeID` as described in Appendix A, *Attributes*.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

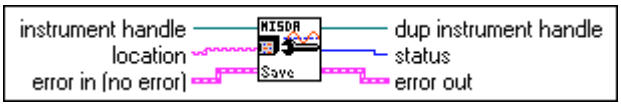
niSda_SaveSetup

Format

C Language

rval = `niSda_SaveSetup` **(ViSession instrumentHandle, ViChar location[])**

G Language



NISDA SaveSetup.vi

Purpose

This function saves the current instrument setup to the given **location**.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
location	`ViChar[]`	Input	Name of the location

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_SaveSetup

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

location is the name of the location to save the setup. This is usually the name of a file on a local disk or a reachable network.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

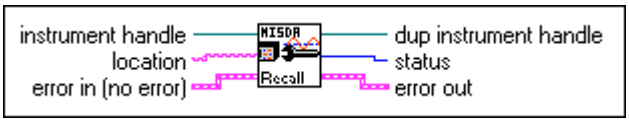
niSda_RecallSetup

Format

C Language

rval = `niSda_RecallSetup` **(ViSession instrumentHandle, ViChar location[])**

G Language



NISDA RecallSetup.vi

Purpose

This function recalls a previously saved instrument setup from the given **location**.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
location	`ViChar[]`	Input	Name of the location

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_RecallSetup

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

location is the name of the location from which to recall the setup of the instrument. This is usually the name of a file on a local disk or a reachable network.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

Utility Functions

The utility functions (`niSda_reset`, `niSda_self_test`, `niSda_error_query`, `niSda_error_message`, and `niSda_revision_query`) allow you to find out if the board is operating correctly, and to reset it if needed. They also allow you to get human-readable information about error conditions while programming the instrument.

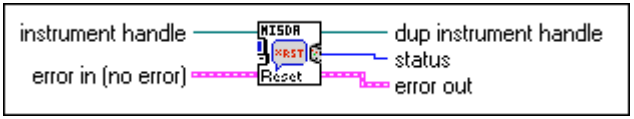
niSda_reset

Format

C Language

rval = `niSda_reset` **(ViSession instrumentHandle)**

G Language



NISDA Reset.vi

Purpose

Places the instrument in a default state. In order to reach the default state, all individual channels are reset, a new configuration is reloaded, and all the instrument attributes are set to their default state as described in Appendix B, *Value Attributes*.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_reset

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

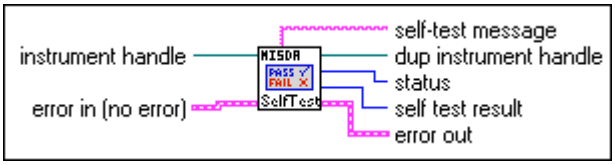
niSda_self_test

Format

C Language

rval = `niSda_self_test` **(ViSession instrumentHandle, ViInt16 *selfTestResult, ViChar selfTestMessage[])**

G Language



NISDA Self Test.vi

Purpose

This function causes the instrument to perform a self-test and returns the result of that self-test. The self-test consists of an internal loop-back. You can load a special self-test configuration to the instrument during the self-test. After the self-test is successfully completed, the instrument is reset to the default state for the current configuration.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
***selfTestResult**	`ViInt16`	Output	Numeric result from self-test operation
selfTestMessage	`ViChar[]`	Output	Self-test status message

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_self_test

continued

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

***selfTestResult** is the numeric result from self-test operation with 0 = no error (test passed)**.**

selfTestMessage is the self-test status message. This string describes in user-readable form the results of the test. It is assumed by the VXI *Plug&Play* specification that this string is no longer than 256 bytes.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

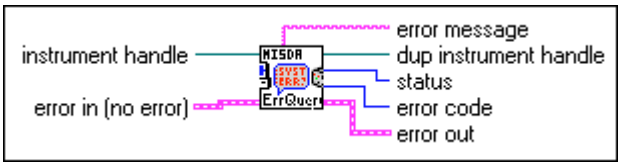
niSda_error_query

Format

C Language

rval = `niSda_error_query` **(ViSession instrumentHandle, ViInt32 *errorCode, ViChar errorMessage[])**

G Language



NISDA Error Query.vi

Purpose

This function queries the instrument and returns instrument-specific error information.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
***errorCode**	`ViInt32`	Output	Instrument error code
errorMessage	`ViChar[]`	Output	Error message

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_error_query

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

***errorCode** is an instrument error code. Please refer to Appendix C, *Status Codes*, for a list of valid error codes.

errorMessage is an Error message. This is a user-readable string that explains the error code. It is assumed by the VXI *Plug&Play* specification that this string is no longer than 256 bytes.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

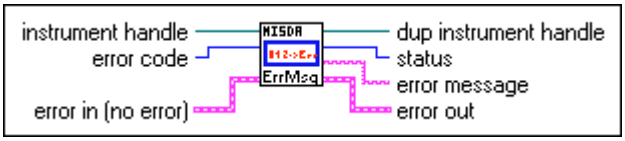
niSda_error_message

Format

C Language

rval = `niSda_error_message` **(ViSession instrumentHandle, ViStatus errorCode, ViChar errorMessage[])**

G Language



NISDA Error Message.vi

Purpose

This function translates the error return value from the SDA instrument driver function to a user-readable string.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
errorCode	`ViStatus`	Input	Instrument driver error code
errorMessage	`ViChar[256]`	Output	SDA instrument driver error message (max 256 characters)

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_error_message

continued

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init. This parameter can be VI_NULL to check error codes returned by a failed niSda_init.

errorCode is an instrument driver error code.

errorMessage is an SDA instrument driver error message in user-readable form. It is assumed by the VXI *Plug&Play* specification that this string is no longer than 256 bytes.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

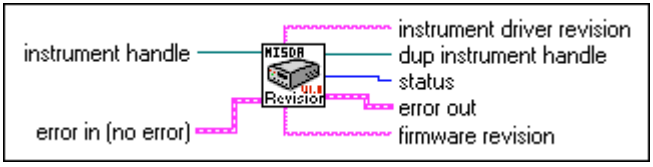
niSda_revision_query

Format

C Language

rval = `niSda_revision_query` **(ViSession instrumentHandle, ViChar instrumentDriverRevision[], ViChar firmwareRevision[])**

G Language



NISDA Revision Query.vi

Purpose

Returns the revision of the instrument driver and the firmware revision of the instrument being used.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
instrumentDriverRevision	`ViChar[256]`	Output	Instrument driver revision in user-readable form
firmwareRevision	`ViChar[256]`	Output	Instrument firm ware revision in user-readable form

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_revision_query

continued

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

instrumentDriverRevision is the instrument driver revision in user-readable form. It is assumed by the VXI *Plug&Play* specification that this string is no longer than 256 bytes.

firmwareRevision is the instrument firm ware revision in user-readable form. It is assumed by the VXI *Plug&Play* specification that this string is no longer than 256 bytes.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

Channel Functions

Channel functions include niSda_ChannelReset, niSda_ChannelConfigureGP, niSda_ChannelConfigureRS232, niSda_ChannelConfigureRS485, niSda_ChannelSetAttributeViBoolean, niSda_ChannelSetAttributeViInt32, niSda_ChannelSetAttributeViReal64, niSda_ChannelSetAttributeViSession, niSda_ChannelSetAttributeViString, niSda_ChannelGetAttributeViBoolean, niSda_ChannelGetAttributeViInt32, niSda_ChannelGetAttributeViReal64, niSda_ChannelGetAttributeViSession, niSda_ChannelGetAttributeViString, niSda_ChannelEnableDataTransceiver, niSda_ChannelDisableDataTransceiver, niSda_ChannelEnableClockTransceiver, niSda_ChannelDisableClockTransceiver, niSda_ChannelReceive, niSda_ChannelReceiveFile, niSda_ChannelCheckReception, niSda_ChannelTransmit, niSda_ChannelTransmitPattern, niSda_ChannelTransmitFile, niSda_ChannelCheckTransmission, niSda_ChannelInsertError, and niSda_ChannelAbort, niSda_ChannelTrigger.

The functions described in this section have channel-wide scope. You can view a channel as a logical data engine that has transceivers and triggers associated with it. Use these functions to configure and control a particular channel. This includes setting and getting the values of particular channel attributes, as well as transmitting and receiving data on a particular channel. Through these functions, you can control the transceivers each channel uses, as well as the triggers by which they are triggered, or the triggers that are sent by each of the channels.

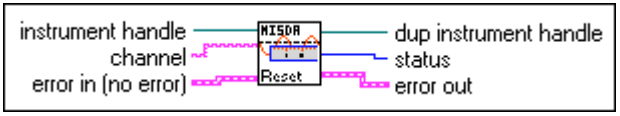
niSda_ChannelReset

Format

C Language

rval = `niSda_ChannelReset` **(ViSession instrumentHandle, ViString channel)**

G Language



NISDA ChannelReset.vi

Purpose

This function places the given channel in the default state. The channel default state is reached by aborting any transmission or reception, emptying all incoming and outgoing buffers and queues while discarding its data, and setting channel attributes to their default values.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_ChannelReset

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

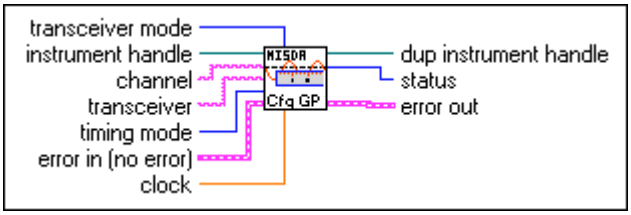
niSda_ChannelConfigureGP

Format

C Language

rval = `niSda_ChannelConfigureGP` **(ViSession instrumentHandle, ViString channel, ViString transceiver, ViInt32 timingMode, ViReal64 clock, ViInt32 transceiverMode)**

G Language



NISDA ChannelConfigureGP.vi

Purpose

This helper function sets channel attributes frequently used in applications that use the general-purpose transceivers.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
transceiver	`ViString`	Input	Transceiver
timingMode	`ViInt32`	Input	Timing mode
clock	`ViReal64`	Input	Clock
transceiverMode	`ViInt32`	Input	Transceiver mode

niSda_ChannelConfigureGP

continued

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

transceiver is the value of the NISDA_CHANNEL_ATTR_XCEIVER attribute.

timingMode is the value of the NISDA_CHANNEL_TIMING_MODE attribute.

clock is the value of the NISDA_CHANNEL_ATTR_CLOCK attribute.

transceiverMode is the value of NISDA_CHANNEL_ATTR_XCEIVER_MODE attribute.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

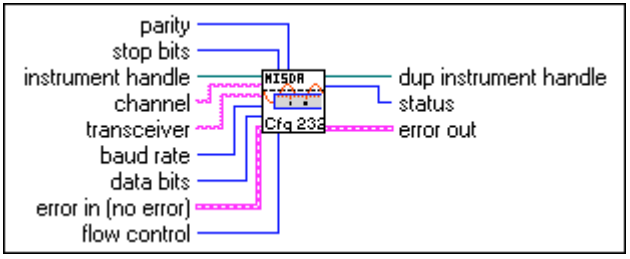
niSda_ChannelConfigureRS232

Format

C Language

rval = `niSda_ChannelConfigureRS232` **(ViSession instrumentHandle, ViString channel, ViString transceiver, ViInt32 baudRate, ViInt32 dataBits, ViInt32 parity, ViInt32 stopBits, ViInt32 flowControl)**

G Language



NISDA ChannelConfigureRS232.vi

Purpose

This helper function sets channel attributes frequently used in applications that use the RS-232 transceivers.

niSda_ChannelConfigureRS232

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
transceiver	`ViString`	Input	Transceiver
baudRate	`ViInt32`	Input	The asynchronous mode baud rate
dataBits	`ViInt32`	Input	The asynchronous mode data bits
parity	`ViInt32`	Input	The asynchronous mode parity
stopBits	`ViInt32`	Input	The asynchronous mode stop bits
flowControl	`ViInt32`	Input	The asynchronous mode flow control

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

niSda_ChannelConfigureRS232

continued

transceiver is the value of the NISDA_CHANNEL_ATTR_XCEIVER attribute.

baudRate is the value of the NISDA_CHANNEL_ATTR_ASRL_BAUD attribute.

dataBits is the value of the NISDA_CHANNEL_ATTR_ASRL_DATA_BITS attribute.

parity is the value of the NISDA_CHANNEL_ATTR_ASRL_PARITY attribute.

stopBits is the value of the NISDA_CHANNEL_ATTR_ASRL_STOP_BITS attribute.

flowControl is the value of the NISDA_CHANNEL_ATTR_ASRL_FLOW_CONTROL attribute.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

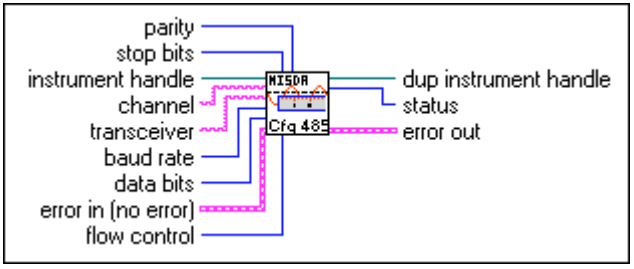
niSda_ChannelConfigureRS485

Format

C Language

rval = `niSda_ChannelConfigureRS485` **(ViSession instrumentHandle, ViString channel, ViString transceiver, ViInt32 baudRate, ViInt32 dataBits, ViInt32 parity, ViInt32 stopBits, ViInt32 flowControl)**

G Language



NISDA ChannelConfigureRS485.vi

Purpose

This helper function sets channel attributes frequently used in applications that use the RS-485 transceivers.

niSda_ChannelConfigureRS485

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	ViSession	Input	Instrument handle
channel	ViString	Input	Channel name
transceiver	ViString	Input	Transceiver
baudRate	ViInt32	Input	The asynchronous mode baud rate
dataBits	ViInt32	Input	The asynchronous mode data bits
parity	ViInt32	Input	The asynchronous mode parity
stopBits	ViInt32	Input	The asynchronous mode stop bits
flowControl	ViInt32	Input	The asynchronous mode flow control

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

niSda_ChannelConfigureRS485

continued

transceiver is the value of the NISDA_CHANNEL_ATTR_XCEIVER attribute.

baudRate is the value of the NISDA_CHANNEL_ATTR_ASRL_BAUD attribute.

dataBits is the value of the NISDA_CHANNEL_ATTR_ASRL_DATA_BITS attribute.

parity is the value of the NISDA_CHANNEL_ATTR_ASRL_PARITY attribute.

stopBits is the value of the NISDA_CHANNEL_ATTR_ASRL_STOP_BITS attribute.

flowControl is the value of the NISDA_CHANNEL_ATTR_ASRL_FLOW_CONTROL attribute.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

niSda_ChannelSetAttribute

Format

C Language

rval = `niSda_ChannelSetAttributeViBoolean` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViBoolean value)**

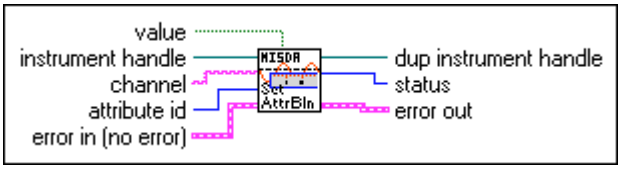
rval = `niSda_ChannelSetAttributeViInt32` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViInt32 value)**

rval = `niSda_ChannelSetAttributeViReal64` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViReal64 value)**

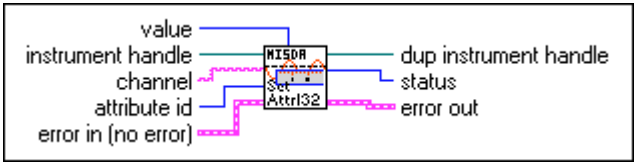
rval = `niSda_ChannelSetAttributeViSession` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViSession value)**

rval = `niSda_ChannelSetAttributeViString` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViChar value[])**

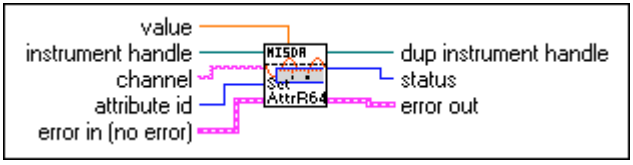
G Language



NISDA ChannelSetAttributeBoolean.vi



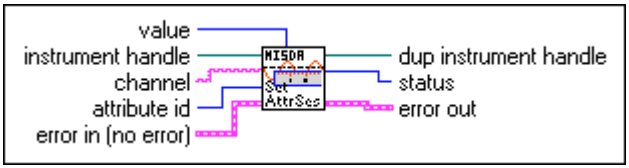
NISDA ChannelSetAttributeInt32.vi



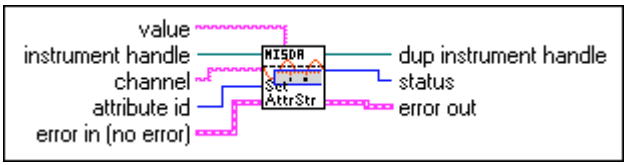
NISDA ChannelSetAttributeReal64.vi

niSda_ChannelSetAttribute

continued



NISDA ChannelSetAttributeSession.vi



NISDA ChannelSetAttributeString.vi

Purpose

This function sets a channel attribute of type `ViBoolean`, `ViInt32`, `ViReal64`, `ViSession`, or `ViChar[]`.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
attributeID	`ViAttr`	Input	Channel attribute

value	`ViBoolean`
	`ViInt32`
	`ViReal64`
	`ViSession`
	`ViChar[]`
		Input	State of the channel attribute

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_ChannelSetAttribute

continued

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

attributeID is the channel attribute for which the state is to be modified. For the list of channel attributes see Appendix A, *Attributes*.

value is the state of the channel attribute. The valid values depend on the value of attributeID as described in Appendix A, *Attributes*.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

niSda_ChannelGetAttribute

Format

C Language

rval = `niSda_ChannelGetAttributeViBoolean` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViBoolean *value)**

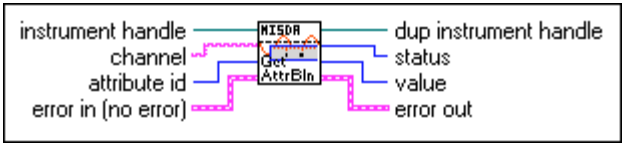
rval = `niSda_ChannelGetAttributeViInt32` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViInt32 *value)**

rval = `niSda_ChannelGetAttributeViReal64` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViReal64 *value)**

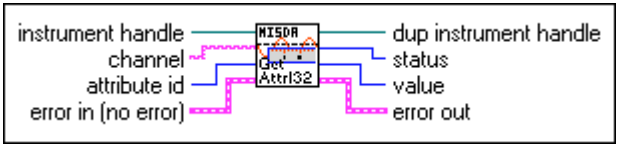
rval = `niSda_ChannelGetAttributeViSession` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViSession *value)**

rval = `niSda_ChannelGetAttributeViString` **(ViSession instrumentHandle, ViString channel, ViAttr attributeID, ViInt32 bufferSize, ViChar value[])**

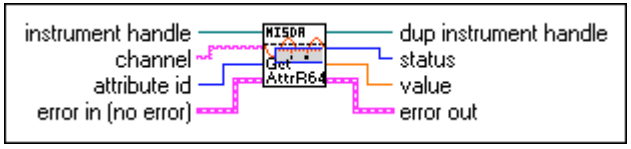
G Language



NISDA ChannelGetAttributeBoolean.vi



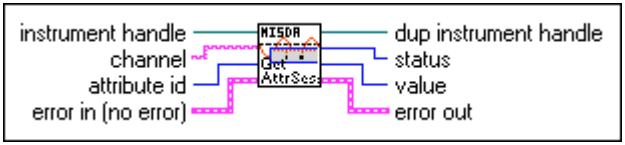
NISDA ChannelGetAttributeInt32.vi



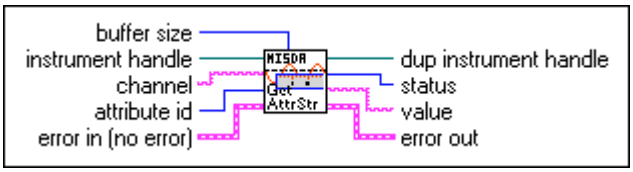
NISDA ChannelGetAttributeReal64.vi

niSda_ChannelGetAttribute

continued



NISDA ChannelGetAttributeSession.vi



NISDA ChannelGetAttributeString.vi

Purpose

This function gets a channel attribute of type `ViBoolean`, `ViInt32`, `ViReal64`, `ViSession`, or `ViChar[]`.

niSda_ChannelGetAttribute

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
attributeID	`ViAttr`	Input	Channel attribute

***value**	`ViBoolean`
	`ViInt32`
	`ViReal64`
	`ViSession`
	`ViChar[]`
		Input	State of the channel attribute

bufferSize	`ViInt32`	Input	Number of characters of the buffer

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

attributeID is the channel attribute for which the state is to be modified. For the list of channel attributes see Appendix A, *Attributes*.

niSda_ChannelGetAttribute

continued

***value** is the state of the channel attribute. The valid values depend on the value of `attributeID` as described in Appendix A, *Attributes*.

bufferSize is the number of characters of the buffer in which the string is going to be copied.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

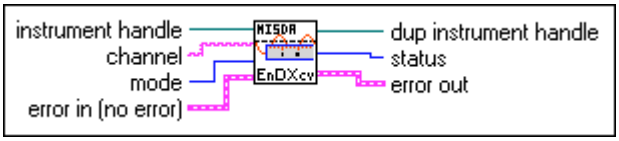
niSda_ChannelEnableDataTransceiver

Format

C Language

rval = `niSda_ChannelEnableDataTransceiver` **(ViSession instrumentHandle, ViString channel, ViInt32 mode)**

G Language



NISDA ChannelEnableDataTransceiver.vi

Purpose

This function enables the data transceiver for input, output or input/output on a given channel depending on the **mode** selected.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_ChannelEnableDataTransceiver

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode can be `NISDA_VAL_OUTPUT`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

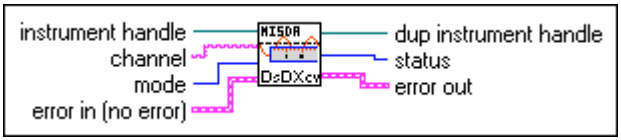
niSda_ChannelDisableDataTransceiver

Format

C Language

rval = `niSda_ChannelDisableDataTranceiver` **(ViSession instrumentHandle, ViString channel, ViInt32 mode)**

G Language



NISDA ChannelDisableDataTransceiver.vi

Purpose

This function disables the data transceiver for input, output or input/output on a given channel depending on the **mode** selected.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_ChannelDisableDataTransceiver

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode can be `NISDA_VAL_OUTPUT`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

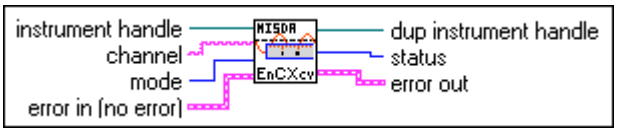
niSda_ChannelEnableClockTransceiver

Format

C Language

rval = `niSda_ChannelEnableClockTranceiver` **(ViSession instrumentHandle, ViString channel, ViInt32 mode)**

G Language



NISDA ChannelEnableClockTransceiver.vi

Purpose

This function enables the clock transceiver for input, output or input/output on a given channel depending on the **mode** selected.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_ChannelEnableClockTransceiver

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode can be `NISDA_VAL_OUTPUT`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

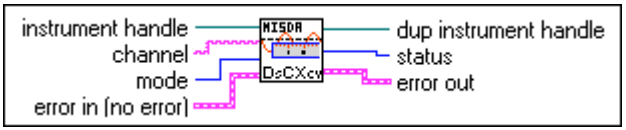
niSda_ChannelDisableClockTransceiver

Format

C Language

rval = `niSda_ChannelDisableClockTranceiver` **(ViSession instrumentHandle, ViString channel, ViInt32 mode)**

G Language



NISDA ChannelDisableClockTransceiver.vi

Purpose

This function disables the clock transceiver for input, output or input/output on a given channel depending on the **mode** selected.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

niSda_ChannelDisableClockTransceiver

continued

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode can be `NISDA_VAL_OUTPUT`.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

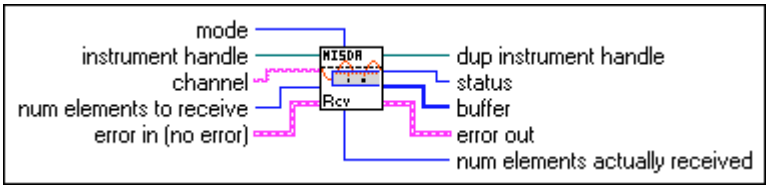
niSda_ChannelReceive

Format

C Language

rval = `niSda_ChannelReceive` **(ViSession instrumentHandle, ViString channel, ViUInt32 numElementsToReceive, ViChar buffer[], ViUInt32 mode, ViUInt32 *numElementsActuallyReceived)**

G Language



NISDA ChannelReceive.vi

Purpose

This function enables the SDA to receive data to a buffer on a given channel. The data can be received in a single shot or continuously, and immediately or deferred, depending on the **mode**. If the **mode** is deferred, the buffer is actually updated when a call to `niSda_ChannelCheckReception` is made. Otherwise, the buffer is updated immediately. If the **mode** is NISDA_RECVMODE_SINGLE, a single buffer is filled; but if the NISDA_RECVMODE_CONTINUOUS is selected, the buffer is constantly updated, until a call to `niSda_ChannelCheckReception` is made with **mode** NISDA_VAL_STOP_RECEPTION.

niSda_ChannelReceive

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
numElementToReceive	`ViUInt32`	Input	Number of elements to receive and maximum size of the buffer
buffer	`ViChar[]`	Input	Pointer to buffer of data
mode	`ViUInt32`	Input	Mode
***numElementsActuallyReceived**	`ViUInt32`	Output	Number of elements actually received

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

niSda_ChannelReceive

continued

numElementToReceive is the number of elements to receive and maximum size of the buffer. The default element size is a bit.

buffer is a pointer to buffer of data. The data is assumed to be aligned to a minimum processor addressable unit.

mode is the mode of reception and can be either NISDA_RECVMODE_IMMEDIATE or NISDA_RECVMODE_DEFERRED, and NISDA_RECVMODE_SINGLE NISDA_RECVMODE_CONTINUOUS.

***numElementsActuallyReceived** is the number of elements actually received.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

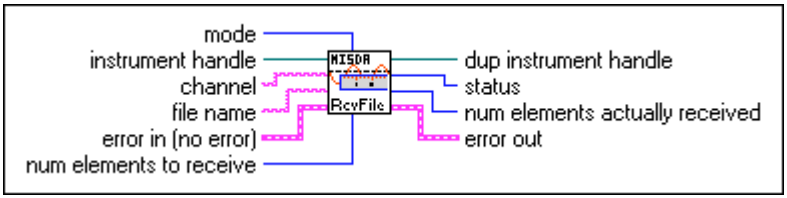
niSda_ChannelReceiveFile

Format

C Language

rval = `niSda_ChannelReceiveFile` **(ViSession instrumentHandle, ViString channel, ViUInt32 numElementsToReceive, ViChar fileName[], ViUInt32 mode, ViUInt32 *numElementsActuallyReceived)**

G Language



NISDA ChannelReceiveFile.vi

Purpose

This function enables the SDA to receive data to a file on a given channel. The data can be received in a single shot or continuously, and immediately or deferred, depending on the **mode**. If the **mode** is deferred, the buffer is actually updated when a call to `niSda_ChannelCheckReception` is made. Otherwise, the buffer is updated immediately. If the **mode** is NISDA_RECVMODE_SINGLE, a single buffer is filled; but if the NISDA_RECVMODE_CONTINUOUS is selected, the buffer is constantly updated, until a call to `niSda_ChannelCheckReception` is made with **mode** NISDA_VAL_STOP.

niSda_ChannelReceiveFile

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
numElementsToReceive	`ViUInt32`	Input	Number of elements to receive and maximum size of the buffer
fileName	`ViChar[]`	Input	Name of the file
mode	`ViUInt32`	Input	Mode
***numElementsActuallyReceived**	`ViUInt32[]`	Output	Number of elements actually received

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

niSda_ChannelReceiveFile

continued

numElementsToReceive is the number of elements to receive and maximum size of the buffer. The default element size is a bit.

fileName name of the file.

mode is the mode of reception and can be either NISDA_RECVMODE_IMMEDIATE or NISDA_RECVMODE_DEFERRED, and NISDA_RECVMODE_SINGLE or NISDA_RECVMODE_CONTINUOUS.

***numElementsActuallyReceived** is the number of elements actually received.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

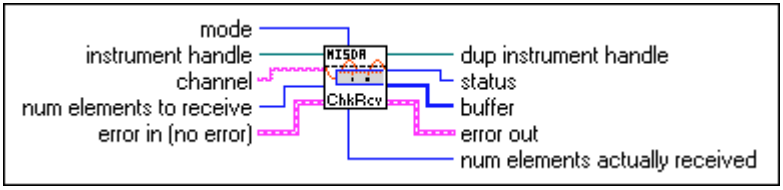
niSda_ChannelCheckReception

Format

C Language

rval = `niSda_ChannelCheckReception` **(ViSession instrumentHandle, ViString channel, ViUInt32 numElementsToReceive, ViChar buffer[], ViUInt32 mode, ViUInt32 *numElementsActuallyReceived)**

G Language



NISDA ChannelCheckReception.vi

Purpose

This function checks the progress of the last current reception on a given channel. Data is transferred to the buffer specified by the original `niSda_ChannelReceive` call, and the number of elements transferred is returned in `numElementsActuallyReceived`. If the **mode** is NISDA_VAL_STOP, the transfer is stop after this call.

niSda_ChannelCheckReception

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
numElementsToReceive	`ViUInt32`	Input	Number of elements to receive and maximum size of the buffer
buffer	`ViChar[]`	Input	Pointer to buffer of data
mode	`ViUInt32`	Input	Mode
***numElementsActuallyReceived**	`ViUInt32[]`	Output	Number of elements actually received

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

niSda_ChannelCheckReception

continued

numElementToReceive is the number of elements to receive and maximum size of the buffer. The default element size is a bit.

buffer is a pointer to buffer of data. The data is assumed to be aligned to a minimum processor addressable unit.

mode valid values are NISDA_VAL_NULL, and NISDA_VAL_STOP. Also refer to modes in the NISDA_CHANNEL_RECEIVE and NISDA_CHANNEL_RECEIVE_FILE operations.

***numElementsActuallyReceived** is the number of elements actually received.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

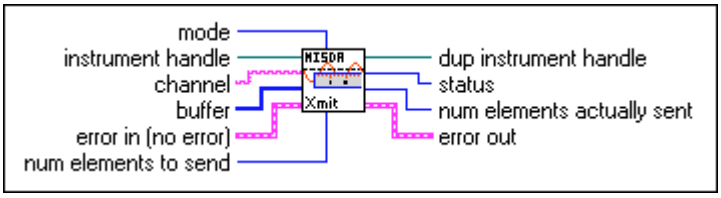
niSda_ChannelTransmit

Format

C Language

rval = `niSda_ChannelTransmit` **(ViSession instrumentHandle, ViString channel, ViChar buffer[], ViUInt32 numElementsToSend, ViUInt32 mode, ViUInt32 *numElementsActuallySent)**

G Language



NISDA ChannelTransmit.vi

Purpose

This function enables the SDA to transmit data from a buffer on a given channel. The data can be transmitted in a single shot or continuously, and immediately or deferred, depending on the **mode**. If the **mode** is deferred, the buffer is actually updated when a call to `niSda_ChannelCheckTransmission` is made. Otherwise, the buffer is updated immediately. If the **mode** is NISDA_TRANSMITMODE_SINGLE, a single buffer is filled; but if the NISDA_TRANSMITMODE_CONTINUOUS is selected, the buffer is constantly updated, until a call to `niSda_ChannelCheckTransmission` is made with **mode** NISDA_VAL_STOP.

niSda_ChannelTransmit

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
buffer	`ViChar[]`	Input	Pointer to the buffer
numElementsToSend	`ViUInt32`	Input	Number of elements to send
mode	`ViUInt32`	Input	Mode
***numElementsActuallySent**	`ViUInt32[]`	Output	Number of elements actually sent

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

buffer is a pointer to the buffer.

niSda_ChannelTransmit

continued

numElementsToSend is the number of elements to send.

mode is the mode of transmission and can be either NISDA_VAL_TRANSMIT_IMMEDIATE or NISDA_VAL_TRANSMIT_DEFERRED, and NISDA_VAL_TRANSMIT_SINGLE or NISDA_VAL_TRANSMIT_CONTINOUS.

***numElementsActuallySent** is the number of elements actually sent.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

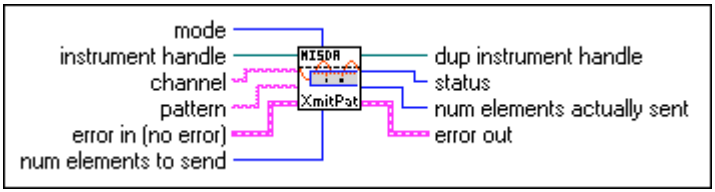
niSda_ChannelTransmitPattern

Format

C Language

rval = `niSda_ChannelTransmitPattern` **(ViSession instrumentHandle, ViString channel, ViChar pattern[], ViUInt32 numElementsToSend, ViUInt32 mode, ViUInt32 *numElementsActuallySent)**

G Language



NISDA ChannelTransmitPattern.vi

Purpose

This function enables the SDA to transmit a given data pattern on a given channel. The data can be transmitted in a single shot or continuously, and immediately or deferred, depending on the `mode`. If the **mode** is deferred, the buffer is actually updated when a call to `niSda_ChannelCheckTransmission` is made. Otherwise, the buffer is updated immediately. If the **mode** is NISDA_TRANSMITMODE_SINGLE, a single buffer is filled; but if the NISDA_TRANSMITMODE_CONTINUOUS is selected, the buffer is constantly updated, until a call to `niSda_ChannelCheckTransmission` is made with **mode** NISDA_VAL_STOP.

niSda_ChannelTransmitPattern

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
pattern	`ViChar[]`	Input	Name of pattern
numElementsToSend	`ViUInt32`	Input	Number of elements to send
mode	`ViUInt32`	Input	Mode
***numBytesActuallySent**	`ViUInt32[]`	Output	Number of bytes actually sent

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

pattern is the name of the pattern.

niSda_ChannelTransmitPattern

continued

numElementsToSend is the number of elements to send.

mode is the mode of transmission and can be either NISDA_VAL_TRANSMIT_IMMEDIATE or NISDA_VAL_TRANSMIT_DEFERRED, and NISDA_VAL_TRANSMIT_SINGLE or NISDA_VAL_TRANSMIT_CONTINOUS.

***numElementsActuallySent** is the number of elements actually sent.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

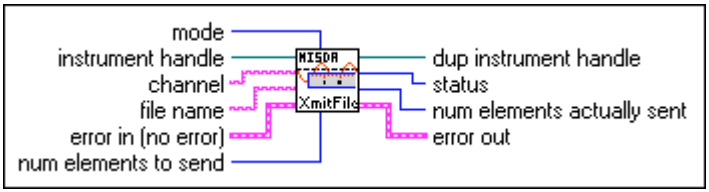
niSda_ChannelTransmitFile

Format

C Language

rval = `niSda_ChannelTransmitFile` **(ViSession instrumentHandle, ViString channel, ViChar fileName[], ViUInt32 numElementsToSend, ViUInt32 mode, ViUInt32 *numElementsActuallySent)**

G Language



NISDA ChannelTransmitFile.vi

Purpose

This function enables the SDA to transmit data from a file on a given channel. The data can be transmitted in a single shot or continuously, and immediately or deferred, depending on the **mode**. If the **mode** is deferred, the buffer is actually updated when a call to `niSda_ChannelCheckTransmission` is made. Otherwise, the buffer is updated immediately. If the **mode** is NISDA_TRANSMITMODE_SINGLE, a single buffer is filled; but if the NISDA_TRANSMITMODE_CONTINUOUS is selected, the buffer is constantly updated, until a call to `niSda_ChannelCheckTransmission` is made with **mode** NISDA_VAL_STOP.

niSda_ChannelTransmitFile

continued

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
fileName	`ViChar[]`	Input	Name of file
numElementsToSend	`ViUInt32`	Input	Number of elements to send
mode	`ViUInt32`	Input	Mode
***numBytesActuallySent**	`ViUInt32[]`	Output	Number of bytes actually sent

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

fileName is the name of file.

niSda_ChannelTransmitFile

continued

numElementsToSend is the number of elements to send.

mode is the mode of transmission and can be either NISDA_VAL_TRANSMIT_IMMEDIATE or NISDA_VAL_TRANSMIT_DEFERRED, and NISDA_VAL_TRANSMIT_SINGLE or NISDA_VAL_TRANSMIT_CONTINOUS.

***numElementsActuallySent** is the number of elements actually sent.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

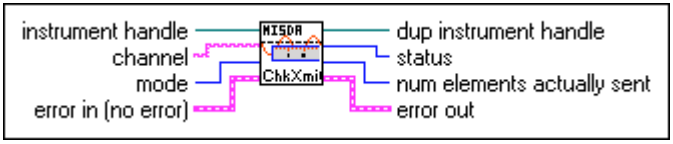
niSda_ChannelCheckTransmission

Format

C Language

rval = `niSda_ChannelCheckTransmission` **(ViSession instrumentHandle, ViString channel, ViUInt32 mode, ViUInt32 *numElementsActuallySent)**

G Language



NISDA ChannelCheckTransmission.vi

Purpose

This function checks the progress of the current transmission on a given channel. Data is transferred from the buffer specified by the original `niSda_ChannelReceive` call, and the number of elements transferred is returned in `numElementsActuallySent`. If the **mode** is NISDA_VAL_STOP, the transfer is stop after this call.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViUInt32`	Input	Mode
***numElementsActuallySent**	`ViUInt32[]`	Output	Number of elements actually sent

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_ChannelCheckTransmission

continued

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up.

mode valid values are NISDA_VAL_NULL, and NISDA_VAL_STOP. Also refer to modes in the NISDA_CHANNEL_TRANSMIT, NISDA_CHANNEL_TRANSMIT_FILE, and NISDATRANSMIT_PATTERN operations.

***numElementsActuallySent** is the number of elements actually sent.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

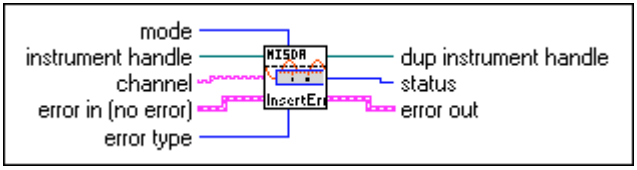
niSda_ChannelInsertError

Format

C Language

rval = `niSda_ChannelInsertError` **(ViSession instrumentHandle, ViString channel, ViUInt32 errorType, ViUInt32 mode)**

G Language



NISDA ChannelInsertError.vi

Purpose

This function inserts an error of type errorType onto a given channel.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
errorType	`ViUInt32`	Input	Error type
mode	`ViUInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_ChannelInsertError

continued

Return Value

Type	Description
`ViStatus`	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from `niSda_init`.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

errorType is Error type NISDA_VAL_ERROR_INVERT, NISDA_VAL_ERROR_STUCK_HIGH, NISDA_VAL_ERROR_STUCK_LOW.

mode is the current valid values for **mode**, which is only NISDA_VAL_NULL.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

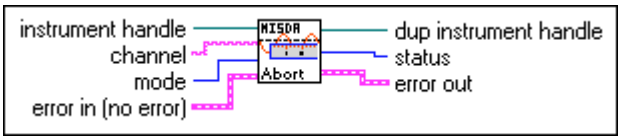
niSda_ChannelAbort

Format

C Language

rval = `niSda_ChannelAbort` **(ViSession instrumentHandle, ViString channel, ViUInt32 mode)**

G Language



NISDA ChannelAbort.vi

Purpose

This function aborts the last reception or transmission on a given channel. If the value of **mode** is NISDA_VAL_ABORT_SYNC, the transfer is aborted in the next logical break point of the protocol being used. If the value of **mode** is NISDA_VAL_ABORT_IMMEDIATE, the transfer is aborted immediately, regardless of the protocol, or the state of the other device.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViUInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_ChannelAbort

continued

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode the valid values for **mode** are NISDA_VAL_ABORT_SYNC, NISDAL_VAL_ABORT_IMMEDIATE.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

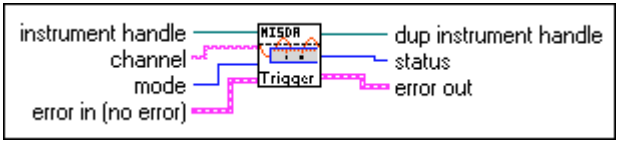
niSda_ChannelTrigger

Format

C Language

rval = `niSda_ChannelTrigger` **(ViSession instrumentHandle, ViString channel, ViUInt32 mode)**

G Language



NISDA ChannelTrigger.vi

Purpose

This function triggers a previously armed condition on a given channel. Depending on the value of **mode**, the trigger is sent synchronous or asynchronous to the instrument clock.

Parameters

Name	Type	Direction	Description
instrumentHandle	`ViSession`	Input	Instrument handle
channel	`ViString`	Input	Channel name
mode	`ViUInt32`	Input	Mode

Error in and error out refer to LabVIEW error cluster input and output. Dup instrument handle is the outgoing duplicate of the incoming instrument handle.

niSda_ChannelTrigger

continued

Return Value

Type	Description
ViStatus	Operational return status

Parameter and Return Value Discussion

instrumentHandle is an instrument handle. This parameter is the value returned from niSda_init.

channel is a string describing the name of the channel. String numbers are always mapped to the corresponding physical channel. Names for the different channels in the configuration utility can also be set up. The strings “0” and “1” represent the first and second channels respectively.

mode the valid values are NISDA_VAL_TRIG_SYNC, and NISDA_VAL_TRIG_ASYNC.

In LabVIEW, the status value returns through the status terminal.

rval returns the following status codes:

NISDA_SUCCESS	operation completed successfully
NISDA_ERROR_INV_OBJECT	the given session reference is invalid

You can find other values that may be returned from this operation in Appendix C, *Status Codes*.

Attributes

Appendix A

This appendix describes the attributes used by the SDA instrument driver. To set/get the value of a given attribute, use the `niSda_Set/GetAttribute`*x* or `niSda_ChannelSet/GetAttribute`*x* operation, for board and channel attributes, respectively, where *x* denotes the type given in the second column of the following table.

☞ **Note:** ***In the following table, the literal string*** `NISDA_` ***precedes the attribute name.***

Name	Data Type	Access	Description
ATTR_NUM_CHANNELS	ViInt32	RO	Returns the number channels
ATTR_RESOURCE_DESCRIPTOR	ViString	RO	The string that describes how to find the physical instrument; from the configuration file
ATTR_IO_SESSION	ViSession	RO	The I/O session or handle used to communicate with the actual instrument
ATTR_VISA_RM_SESSION	ViSession	RO	The VISA resource manager session used to open instrument I/O sessions
ATTR_DRIVER_MAJOR_VERSION	ViInt32	RO	Specifies the major version number of the specific instrument driver. Set by the specific driver
ATTR_DRIVER_MINOR_VERSION	ViInt32	RO	Specifies the minor version number of the specific instrument driver. Set by the specific driver
ATTR_CLASS_MAJOR_VERSION	ViInt32	RO	Specifies the class driver major version number. Set by the class
ATTR_CLASS_MINOR_VERSION	ViInt32	RO	Specifies the class driver minor version number. Set by the class
ATTR_DRIVER_REVISION	ViString	RO	A string that gives additional version information about the instrument driver. Set by the specific driver

Name	Data Type	Access	Description
ATTR_CLASS_REVISION	ViString	RO	A String that gives additional version information about the class instrument driver
ATTR_USER_DATA	ViInt32	R/W	User data
ATTR_SLOT	ViInt32	R/W	Physical location of the instrument
ATTR_MODEL_CODE	ViInt32	R/W	The model code of this instrument
ATTR_MANF_ID	ViInt32	R/W	The manufacturer Id (National Instruments) of this instrument
CHANNEL_ATTR_MAX_QUEUE_LENGTH	ViInt32	R/W	Length of the channel data queue
CHANNEL_ATTR_TMO_VALUE	ViInt32	R/W	Timeout value for transmission and reception in milliseconds
CHANNEL_ATTR_ASRL_BAUD	ViInt32	R/W	The asynchronous mode baud rate
CHANNEL_ATTR_ASRL_DATA_BITS	ViInt32	R/W	The asynchronous mode data bits
CHANNEL_ATTR_ASRL_PARITY	ViInt32	R/W	The asynchronous mode parity
CHANNEL_ATTR_ASRL_STOP_BITS	ViInt32	R/W	The asynchronous mode stop bits
CHANNEL_ATTR_ASRL_FLOW_CNTRL	ViInt32	R/W	The asynchronous mode flow control
CHANNEL_ATTR_TRANSFER_ELEMENT_SIZE	ViInt32	R/W	Element size used in any data transfer Defined Values: NISDA_VAL_BIT NISDA_VAL_BYTE Or any other addressable by the computer on which the application is running
CHANNEL_ATTR_TRIGGER_SOURCE	ViInt32	R/W	Extended values for `TRIGGER_SOURCE`. Defined Values: NISDA_VAL_IMMEDIATE NISDA_VAL_EXTERNAL NISDA_VAL_BUS
CHANNEL_ATTR_XCEIVER	ViString	R/W	Transceiver
CHANNEL_ATTR_XCEIVER_MODE	ViInt32	R/W	Transceiver mode
CHANNEL_ATTR_FRAMING_MODE	ViInt32	R/W	Framing mode

Name	Data Type	Access	Description
CHANNEL_ATTR_TRIG_COND	ViInt32	R/W	Trigger condition function
CHANNEL_ATTR_TRIG_COND_A	ViInt32	R/W	Trigger condition A
CHANNEL_ATTR_TRIG_COND_B	ViInt32	R/W	Trigger condition B
CHANNEL_ATTR_TRIG_CONTROL	ViInt32	R/W	Trigger control information
CHANNEL_ATTR_TRIG_ROUTE	ViInt32	R/W	Trigger routing information
CHANNEL_ATTR_TRIG_PRECNTR	ViInt32	R/W	Pretrigger counter
CHANNEL_ATTR_TRIG_POSTCNTR	ViInt32	R/W	Posttrigger counter
CHANNEL_ATTR_TRIG_PATTERN_MATCH_63_32	ViInt32	R/W	Pattern to match (high word)
CHANNEL_ATTR_TRIG_PATTERN_MATCH_31_0	ViInt32	R/W	Pattern to match (low word)
CHANNEL_ATTR_TRIG_PATTERN_MASK_63_32	ViInt32	R/W	Pattern mask (high word)
CHANNEL_ATTR_TRIG_PATTERN_MASK_31_0	ViInt32	R/W	Pattern mask (low word)
CHANNEL_ATTR_TIMING_MODE	ViInt32	R/W	Timing mode
CHANNEL_ATTR_CLOCK	ViInt32	R/W	Clock

Value Attributes

Appendix B

This appendix describes the possible values for each attribute described in Appendix A, *Attributes*. A given value may not be valid in a given state. In this case, the status code from the `SetAttribute` will indicate an invalid condition. See the description of the `SetAttribute` calls for details on the return values.

The following are channel-based SDA instrument driver attributes.

Attributes	Valid Values
ATTR_NUM_CHANNELS	"0", "1" aliased to "A", "B"
ATTR_IO_SESSION	0-0xffffffff
ATTR_VISA_RM_SESSION	0-0xffffffff
ATTR_DRIVER_MAJOR_VERSION	0-0xffffffff
ATTR_DRIVER_MINOR_VERSION	0-0xffffffff
ATTR_CLASS_MAJOR_VERSION	0-0xffffffff
ATTR_CLASS_MINOR_VERSION	0-0xffffffff
ATTR_DRIVER_REVISION	0-0xffffffff
ATTR_CLASS_REVISION	0-0xffffffff
ATTR_USER_DATA	0-0xffffffff
ATTR_SLOT	0-0xffffffff
ATTR_MODEL_CODE	0-0xffffffff
ATTR_MANF_ID	0-0xffffffff
ATTR_FIRMWARE_PROGRAM	String

Attributes	Valid Values
CHANNEL_ATTR_MAX_QUEUE_LENGTH	0-0xffffffff
CHANNEL_ATTR_TMO_VALUE	0-0xffffffff NISDA_TMO_IMMEDIATE NISDA_TMO_INFINITE
CHANNEL_ATTR_ASRL_BAUD	0-0xffffffff
CHANNEL_ATTR_ASRL_DATA_BITS	5-9
CHANNEL_ATTR_ASRL_PARITY	NISDA_VAL_ASRL_PAR_NONE NISDA_VAL_ASRL_PAR_ODD NISDA_VAL_ASRL_PAR_EVEN NISDA_VAL_ASRL_PAR_SPACE NISDA_VAL_ASRL_PAR_MARK
CHANNEL_ATTR_ASRL_STOP_BITS	NISDA_VAL_ASRL_STOP_ONE NISDA_VAL_ASRL_STOP_TWO
CHANNEL_ATTR_ASRL_FLOW_CNTRL	NISDA_VAL_ASRL_FLOW_NONE NISDA_VAL_ASRL_FLOW_XON_XOFF NISDA_VAL_ASRL_FLOW_RTS_CTS
CHANNEL_ATTR_TRANSFER_ELEMENT_SIZE	NISDA_VAL_BIT NISDA_VAL_BYTE
CHANNEL_ATTR_XCEIVER	NISDA_VAL_XCEIVER_PCH0 NISDA_VAL_XCEIVER_PCH1 NISDA_VAL_XCEIVER_RS232CH0 NISDA_VAL_XCEIVER_RS232CH1 NISDA_VAL_XCEIVER_RS485CH0 NISDA_VAL_XCEIVER_RS485CH1
CHANNEL_ATTR_XCEIVER_MODE	NISDA_VAL_XCEIVER_XV NISDA_VAL_XCEIVER_TTL NISDA_VAL_XCEIVER_ECL NISDA_VAL_XCEIVER_3V NISDA_VAL_XCEIVER_RS232
CHANNEL_ATTR_XCEIVER_LV	−10.0 to 10.0
CHANNEL_ATTR_XCEIVER_HV	−10.0 to 10.0

Attributes	Valid Values
CHANNEL_ATTR_XCEIVER_TH	−10.0 To 10.0
CHANNEL_ATTR_FRAMING_MODE	NISDA_VAL_FRAMING_NONE NISDA_VAL_FRAMING_X21
CHANNEL_ATTR_TRIG_COND	NISDA_VAL_TRIG_COND_ALWAYS NISDA_VAL_TRIG_COND_A NISDA_VAL_TRIG_COND_NOTA NISDA_VAL_TRIG_COND_B NISDA_VAL_TRIG_COND_NOTB NISDA_VAL_TRIG_COND_A_OR_B NISDA_VAL_TRIG_COND_A_OR_NOTB NISDA_VAL_TRIG_COND_NOTA_OR_B NISDA_VAL_TRIG_COND_NOTA_OR_NOTB NISDA_VAL_TRIG_COND_A_AND_B NISDA_VAL_TRIG_COND_A_AND_NOTB NISDA_VAL_TRIG_COND_NOTA_AND B NISDA_VAL_TRIG_COND_NOTA_AND_NOTB NISDA_VAL_TRIG_COND_A_XOR_B NISDA_VAL_TRIG_COND_A_XNOR_B NISDA_VAL_TRIG_COND_NEVER
CHANNEL_ATTR_TRIG_COND_A	NISDA_VAL_TRIG_COND_TRIGIN_PIN NISDA_VAL_TRIG_COND_FALL_EDGE NISDA_VAL_TRIG_COND_RISE_EDGE NISDA_VAL_TRIG_COND_ANY_EDGE NISDA_VAL_TRIG_COND_PATTERN_MATCH NISDA_VAL_TRIG_COND_SERIAL_PIN NISDA_VAL_TRIG_COND_OTHER_SLOT
CHANNEL_ATTR_TRIG_COND_B	NISDA_VAL_TRIG_COND_TRIGIN_PIN NISDA_VAL_TRIG_COND_FALL_EDGE NISDA_VAL_TRIG_COND_RISE_EDGE NISDA_VAL_TRIG_COND_ANY_EDGE NISDA_VAL_TRIG_COND_PATTERN_MATCH NISDA_VAL_TRIG_COND_SERIAL_PIN NISDA_VAL_TRIG_COND_OTHER_SLOT

Attributes	Valid Values
CHANNEL_ATTR_TRIG_ROUTE	NISDA_VAL_TRIG_ROUTE_AQUIRE NISDA_VAL_TRIG_ROUTE_GENERATE NISDA_VAL_TRIG_ROUTE_TRIGOUT_PIN NISDA_VAL_TRIG_ROUTE_SERIAL_PIN NISDA_VAL_TRIG_ROUTE_OTHER_SLOT
CHANNEL_ATTR_TRIG_POSTCNTR	0-0xffffffff
CHANNEL_ATTR_TRIG_PATTERN_MATCH_63_32	0-0xffffffff
CHANNEL_ATTR_TRIG_PATTERN_MATCH_31_0	0-0xffffffff
CHANNEL_ATTR_TRIG_PATTERN_MASK_63_32	0-0xffffffff
CHANNEL_ATTR_TRIG_PATTERN_MASK_31_0	0-0xffffffff
CHANNEL_ATTR_TIMING_MODE	NISDA_TIMING_SYNC NISDA_TIMING_ASYNC
CHANNEL_ATTR_CLOCK	0-0xffffffff

Status Codes

Appendix C

This appendix describes the status codes returned by the SDA.

Each SDA function returns a status code that indicates whether the function was performed successfully. All error codes are mapped as negative values, while success values are mapped as zero or positive. A summary of the status codes is listed.

Completion Codes	Description
NISDA_WARN_NSUP_ID_QUERY	ID query not supported
NISDA_SUCCESS	Successful Completion
NISDA_ERROR_INV_OBJECT	Invalid object
NISDA_WARN_NSUP_RESET	Reset not supported
NISDA_WARN_NSUP_SELF_TEST	Self-test not supported
NISDA_WARN_NSUP_ERROR_QUERY	Error query not supported
NISDA_WARN_NSUP_REV_QUERY	Revision query not supported
NISDA_ERROR_PARAMETER1	Parameter 1 out of range
NISDA_ERROR_PARAMETER2	Parameter 2 out of range
NISDA_ERROR_PARAMETER3	Parameter 3 out of range
NISDA_ERROR_PARAMETER4	Parameter 4 out of range
NISDA_ERROR_PARAMETER5	Parameter 5 out of range
NISDA_ERROR_PARAMETER6	Parameter 6 out of range
NISDA_ERROR_PARAMETER7	Parameter 7 out of range
NISDA_ERROR_PARAMETER8	Parameter 8 out of range
NISDA_ERROR_FAIL_ID_QUERY	Identification query failed

Completion Codes	Description
NISDA_ERROR_INV_RESPONSE	Error interpreting instrument response
NISDA_ERROR_INSTR_SPECIFIC	Instrument specific error
NISDA_ERROR_FILE_OPEN	File is open
NISDA_ERROR_CANT_OPEN_FILE	Cannot open file
NISDA_ERROR_READING_FILE	Error reading from file
NISDA_ERROR_WRITING_FILE	Error writing to file
NISDA_ERROR_FILE_NOT_FOUND	File not found
NISDA_ERROR_INVALID_FILE_FORMAT	Invalid file format
NISDA_ERROR_INVALID_PATHNAME	Invalid path name
NISDA_ERROR_LOADING_EXTERNAL_MODULE	Failure loading external module
NISDA_ERROR_INVALID_ATTRIBUTE	Invalid attribute
NISDA_ERROR_ATTR_NOT_WRITEABLE	Attribute is not writable
NISDA_ERROR_ATTR_NOT_READABLE	Attribute is not readable
NISDA_ERROR_INVALID_PARAMETER	Invalid parameter
NISDA_ERROR_INVALID_VALUE	Invalid value
NISDA_ERROR_CONFIG_ENTRY_NOT_FOUND	Configuration entry not found
NISDA_ERROR_CONFIG_FILE_NOT_FOUND	Configuration file not found
NISDA_ERROR_FUNCTION_NOT_SUPPORTED	Function not supported
NISDA_ERROR_ATTRIBUTE_NOT_SUPPORTED	Attribute not supported
NISDA_ERROR_VALUE_NOT_SUPPORTED	Value not supported
NISDA_ERROR_INVALID_TYPE	Invalid type
NISDA_ERROR_TYPES_DO_NOT_MATCH	Types do not match
NISDA_ERROR_DEFERRED_VALUE_CONFLICT	The specified attribute already has a value waiting to be updated
NISDA_ERROR_ITEM_ALREADY_EXISTS	The specified item already exists

Completion Codes	Description
NISDA_ERROR_INVALID_CONFIGURATION	Not a valid configuration
NISDA_ERROR_VALUE_NOT_AVAILABLE	The requested item or value does not exist or is not available
NISDA_ERROR_ATTRIBUTE_VALUE_NOT_KNOWN	The requested attribute value not known and cannot be determined
NISDA_ERROR_INVALID_INFO	The requested information is invalid
NISDA_ERROR_NOT_INITIALIZED	Object or item is not initialized

Customer Communication

Appendix D

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

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

Electronic Services

Bulletin Board Support

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

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

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

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

FTP Support

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

Fax-on-Demand Support

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

E-Mail Support (Currently USA Only)

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

`support@natinst.com`

Telephone and Fax Support

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

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

Click here to comment on this document via the National Instruments website at http://www.natinst.com/documentation/daq/

Technical Support Form

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

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

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___) _____

Computer brand _____ Model _____ Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem:_____

_____

_____

_____

Click here to comment on this document via the National Instruments website at http://www.natinst.com/documentation/daq/

Serial Data Analyzer Hardware and Software Configuration Form

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

National Instruments Products

Computer-Based Instrument hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments Application Software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

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

Title: *Serial Data Analyzer Software Reference Manual*

Edition Date: December 1997

Part Number: 321753A-01

Click here to comment on this document via the National Instruments website at http://www.natinst.com/documentation/daq/

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

_____

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (___)_____ Fax (___) _____

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

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Prefix	Meaning	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}

Numbers/Symbols

%	percent
+	positive of, or plus
-	negative of, or minus
/	per
°	degree
Ω	ohm

A

A	amperes
AC	alternating current
AC coupled	allowing the transmission of AC signals while blocking DC signals
A/D	analog-to-digital
ADC	analog-to-digital converter—an electronic device, often an integrated circuit, that converts an analog voltage to a digital number
address	character code that identifies a specific location (or series of locations) in memory
ADE	application development environment—examples of ADE are LabVIEW, LabWindows/CVI, Visual Basic, and Visual C++
amplification	a type of signal conditioning that improves accuracy in the resulting digitized signal and reduces noise
ANSI	American National Standards Institute
API	application programming interface
ASIC	Application-Specific Integrated Circuit—a proprietary semiconductor component designed and manufactured to perform a set of specific functions for a specific customer
asynchronous	(1) hardware—a property of an event that occurs at an arbitrary time, without synchronization to a reference clock (2) software—a property of a function that begins an operation and returns prior to the completion or termination of the operation
attenuate	to decrease the amplitude of a signal
attenuation ratio	the factor by which a signal's amplitude is decreased

B

b	bit—one binary digit, either 0 or 1
B	byte—eight related bits of data, an eight-bit binary number. Also used to denote the amount of memory required to store one byte of data.
bandwidth	the range of frequencies present in a signal, or the range of frequencies to which a measuring device can respond
base address	a memory address that serves as the starting address for programmable registers. All other addresses are located by adding to the base address.
baud rate	serial communications data transmission rate expressed in bits per second (b/s)
BCD	binary-coded decimal
binary	a number system with a base of 2
BIOS	basic input/output system—BIOS functions are the fundamental level of any PC or compatible computer. BIOS functions embody the basic operations needed for successful use of the computer's hardware resources.
bipolar	a signal range that includes both positive and negative values (for example, –5 V to +5 V)
BNC	a type of coaxial signal connector
buffer	temporary storage for acquired or generated data (software)
burst-mode	a high-speed data transfer in which the address of the data is sent followed by back-to-back data words while a physical signal is asserted
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 AT bus, NuBus, Micro Channel, and EISA bus.
bus master	a type of a plug-in board or controller with the ability to read and write devices on the computer bus

C

C	Celsius
cache	high-speed processor memory that buffers commonly used instructions or data to increase processing throughput
channel	pin or wire lead to which you apply or from which you read the analog or digital signal. Analog signals can be single-ended or differential. For digital signals, you group channels to form ports. Ports usually consist of either four or eight digital channels.
channel clock	the clock controlling the time interval between individual channel sampling within a scan. Boards with simultaneous sampling do not have this clock.
circuit trigger	a condition for starting or stopping clocks
clock	hardware component that controls timing for reading from or writing to groups
CMOS	complementary metal-oxide semiconductor
conversion device	device that transforms a signal from one form to another. For example, analog-to-digital converters (ADCs) for analog input, digital-to-analog converters (DACs) for analog output, digital input or output ports, and counter/timers are conversion devices.
conversion time	the time required, in an analog input or output system, from the moment a channel is interrogated (such as with a read instruction) to the moment that accurate data is available
counter/timer	a circuit that counts external pulses or clock pulses (timing)
coupling	the manner in which a signal is connected from one location to another
CPU	central processing unit
crosstalk	an unwanted signal on one channel due to an input on a different channel
current drive capability	the amount of current a digital or analog output channel is capable of sourcing or sinking while still operating within voltage range specifications

current sinking	the ability of an instrument to dissipate current for analog or digital output signals
current sourcing	the ability of an instrument to supply current for analog or digital output signals

D

D/A	digital-to-analog
DAC	digital-to-analog converter—an electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current
DAQ	data acquisition—(1) collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures and inputting them to a computer for processing; (2) collecting and measuring the same kinds of electrical signals with A/D and/or DIO boards plugged into a computer, and possibly generating control signals with D/A and/or DIO boards in the same computer
dB	decibel—the unit for expressing a logarithmic measure of the ratio of two signal levels: dB = 20log10 V1/V2, for signals in volts
DC	direct current
DC coupled	allowing the transmission of both AC and DC signals
device	a plug-in instrument card or pad that can contain multiple channels and conversion devices. Plug-in boards and PCMCIA cards, which connects to your computer parallel port, are examples of devices.
digital trigger	a TTL level signal having two discrete levels—a high and a low level
DIN	Deutsche Industrie Norme
DIO	digital input/output
DMA	direct memory access—a method by which data can be transferred to/from computer memory from/to a device or memory on the bus while the processor does something else. DMA is the fastest method of transferring data to/from computer memory.
down counter	performing frequency division on an internal signal

DRAM	dynamic RAM
drivers	software that controls a specific hardware device such as a plug-in instrument or a GPIB interface board
dual-access memory	memory that can be sequentially accessed by more than one controller or processor but not simultaneously accessed. Also known as shared memory.
dual-ported memory	memory that can be simultaneously accessed by more than one controller or processor

E

ECL	emitter-coupled logic
EEPROM	electrically erasable programmable read-only memory—ROM that can be erased with an electrical signal and reprogrammed
electrostatically coupled	propagating a signal by means of a varying electric field
EMC	electromechanical compliance
EPROM	erasable programmable read-only memory—ROM that can be erased (usually by ultraviolet light exposure) and reprogrammed
event	the condition or state of an analog or digital signal
external trigger	a voltage pulse from an external source that triggers an event such as an A/D conversion

F

false triggering	triggering that occurs at an unintended time
fetch-and-deposit	a data transfer in which the data bytes are transferred from the source to the controller, and then from the controller to the target

FIFO	first-in first-out memory buffer—the first data stored is the first data sent to the acceptor. FIFOs are often used on DAQ devices to temporarily store incoming or outgoing data until that data can be retrieved or output. For example, an analog input FIFO stores the results of A/D conversions until the data can be retrieved into system memory, a process that requires the servicing of interrupts and often the programming of the DMA controller. This process can take several milliseconds in some cases. During this time, data accumulates in the FIFO for future retrieval. With a larger FIFO, longer latencies can be tolerated. In the case of analog output, a FIFO permits faster update rates, because the waveform data can be stored on the FIFO ahead of time. This again reduces the effect of latencies associated with getting the data from system memory to the DAQ device.
filtering	a type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure
flyby	a type of high-performance data transfer in which the data bytes pass directly from the source to the target without being transferred to the controller
ft	feet

G

gain	the factor by which a signal is amplified, sometimes expressed in decibels
GPIB	General Purpose Interface bus, synonymous with HP-IB. The standard bus used for controlling electronic instruments with a computer. Also called IEEE 488 bus because it is defined by ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987.

H

h	hour
handle	pointer to a pointer to a block of memory; handles reference arrays and strings. An array of strings is a handle to a block of memory containing handles to strings.

handler	a device driver that is installed as part of the operating system of the computer
handshaked digital I/O	a type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. Also called latched digital I/O.
hardware	the physical components of a computer system, such as the circuit boards, plug-in boards, chassis, enclosures, peripherals, cables, and so on
hardware triggering	a form of triggering where you set the start time of an acquisition and gather data at a known position in time relative to a trigger signal
hex	hexadecimal
Hz	hertz—the number of scans read or updates written per second

I

IC	integrated circuit
ID	identification
IDE	integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
IEEE 488	the shortened notation for ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987. See also GPIB.
immediate digital I/O	a type of digital acquisition/generation where LabVIEW updates the digital lines or port states immediately or returns the digital value of an input line. Also called nonlatched digital I/O.
in.	inches
Industrial Device Networks	standardized digital communications networks used in industrial automation applications; they often replace vendor-proprietary networks so that devices from different vendors can communicate in control systems
input bias current	the current that flows into the inputs of a circuit

input impedance	the measured resistance and capacitance between the input terminals of a circuit
input offset current	the difference in the input bias currents of the two inputs of an instrumentation amplifier
instrument driver	a set of high-level software functions that controls a specific plug-in, DAQ, PXI, GPIB, VXI, or RS-232 programmable instrument. Instrument drivers are available in several forms, ranging from a function callable language to a virtual instrument (VI) in LabVIEW.
instrumentation amplifier	a circuit whose output voltage with respect to ground is proportional to the difference between the voltages at its two inputs
interrupt	a computer signal indicating that the CPU should suspend its current task to service a designated activity
interrupt level	the relative priority at which a device can interrupt
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
IRQ	interrupt request

K

k	kilo—the standard metric prefix for 1,000, or 10^3, used with units of measure such as volts, hertz, and meters
K	kilo—the prefix for 1,024, or 2^{10}, used with B in quantifying data or computer memory

L

latched digital I/O	a type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. Also called handshaked digital I/O.
LED	light-emitting diode

library	a file containing compiled object modules, each comprised of one of more functions, that can be linked to other object modules that make use of these functions. NISDA.LIB is a library that contains instrument driver functions. The NI-DAQ function set is broken down into object modules so that only the object modules that are relevant to your application are linked in, while those object modules that are not relevant are not linked.
LSB	least significant bit

M

m	meters
M	(1) Mega, the standard metric prefix for 1 million or 10^6, when used with units of measure such as volts and hertz; (2) mega, the prefix for 1,048,576, or 2^{20}, when used with B to quantify data or computer memory
MB	megabytes of memory
MBLT	eight-byte block transfers in which both the Address bus and the Data bus are used to transfer data
Mbytes/s	a unit for data transfer that means 1 million or 10^6 bytes/s
memory buffer	*See* buffer.
MIPS	million instructions per second—the unit for expressing the speed of processor machine code instructions
MS	million samples
MSB	most significant bit
MTBF	mean time between failure

N

NI-SDA	NI instrument driver for SDA cards
NIST	National Institute of Standards and Technology

nodes	execution elements of a block diagram consisting of functions, structures, and subVIs
noise	an undesirable electrical signal—Noise comes from external sources such as the AC power line, motors, generators, transformers, fluorescent lights, soldering irons, CRT displays, computers, electrical storms, welders, radio transmitters, and internal sources such as semiconductors, resistors, and capacitors. Noise corrupts signals you are trying to send or receive.
nonlatched digital I/O	a type of digital acquisition/generation where LabVIEW updates the digital lines or port states immediately or returns the digital value of an input line. Also called immediate digital I/O or non-handshaking.
nonreferenced signal sources	signal sources with voltage signals that are not connected to an absolute reference or system ground. Also called floating signal sources. Some common example of nonreferenced signal sources are batteries, transformers, or thermocouples.

O

onboard channels	channels provided by the plug-in DAQ board
onboard RAM	optional RAM usually installed into SIMM slots
operating system	base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices
optical coupler, optocoupler	a device designed to transfer electrical signals by utilizing light waves to provide coupling with electrical isolation between input and output. Sometimes called optoisolator or photocoupler.
optical isolation	the technique of using an optoelectric transmitter and receiver to transfer data without electrical continuity, to eliminate high-potential differences and transients
OUT	output pin—a counter output pin where the counter can generate various TTL pulse waveforms

output settling time	the amount of time required for the analog output voltage to reach its final value within specified limits
output slew rate	the maximum rate of change of analog output voltage from one level to another

P

pattern generation	a type of handshaked (latched) digital I/O in which internal counters generate the handshaked signal, which in turn initiates a digital transfer. Because counters output digital pulses at a constant rate, this means you can generate and retrieve patterns at a constant rate because the handshaked signal is produced at a constant rate.
PC Card	a credit-card-sized expansion card that fits in a PCMCIA slot, often referred to as a PCMCIA card
PCI	Peripheral Component Interconnect—a high-performance expansion bus architecture originally developed by Intel to replace ISA and EISA. It is achieving widespread acceptance as a standard for PCs and work-stations; it offers a theoretical maximum transfer rate of 132 Mbytes/s.
PCI-MITE	is a custom ASIC designed by National Instruments that implements the PCI bus interface. The PCI-MITE supports bus mastering for high speed data transfers over the PCI bus. It is also used in PXI cards.
PCMCIA	an expansion bus architecture that has found widespread acceptance as a *de facto* standard in notebook-size computers. It originated as a specification for add-on memory cards written by the Personal Computer Memory Card International Association.
pipeline	a high-performance processor structure in which the completion of an instruction is broken into its elements so that several elements can be processed simultaneously from different instructions
Plug and Play devices	devices that do not require DIP switches or jumpers to configure resources on the devices—also called switchless devices
port	(1) a communications connection on a computer or a remote controller (2) a digital port, consisting of four or eight lines of digital input and/or output

posttriggering	the technique used on an instrument to acquire a programmed number of samples after trigger conditions are met
ppm	parts per million
pretriggering	the technique used on an instrument to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition
propagation	the transmission of a signal through a computer system
propagation delay	the amount of time required for a signal to pass through a circuit
protocol	the exact sequence of bits, characters, and control codes used to transfer data between computers and peripherals through a communications channel.
pts	points
pulse trains	multiple pulses
pulsed output	a form of counter signal generation by which a pulse is outputted when a counter reaches a certain value
PXI	stands for PCI eXtensions for Instrumentation. PXI is an open specification that builds off the CompactPCI specification by adding instrumentation-specific features.

R

RAM	random-access memory
real time	a property of an event or system in which data is processed as it is acquired instead of being accumulated and processed at a later time
resolution	the smallest signal increment that can be detected by a measurement system. Resolution can be expressed in bits, in proportions, or in percent of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244 percent of full scale.
retry	an acknowledge by a destination that signifies that the cycle did not complete and should be repeated
ribbon cable	a flat cable in which the conductors are side by side

rise time	the difference in time between the 10% and 90% points of a system's step response
rms	root mean square—the square root of the average value of the square of the instantaneous signal amplitude; a measure of signal amplitude
ROM	read-only memory
RTSI bus	real-time system integration bus—the National Instruments timing bus that connects instruments directly, by means of connectors on top of the boards, for precise synchronization of functions

S

s	seconds
S	samples
sample counter	the clock that counts the output of the channel clock, in other words, the number of samples taken. On boards with simultaneous sampling, this counter counts the output of the scan clock and hence the number of scans.
scan	one or more analog or digital input samples. Typically, the number of input samples in a scan is equal to the number of channels in the input group. For example, one pulse from the scan clock produces one scan which acquires one new sample from every analog input channel in the group.
scan clock	the clock controlling the time interval between scans. On boards with interval scanning support (for example, the AT-MIO-16F-5), this clock gates the channel clock on and off. On boards with simultaneous sampling (for example, the EISA-A2000), this clock clocks the track-and-hold circuitry.
scan rate	the number of scans per second. For example, a scan rate of 10 Hz means sampling each channel 10 times per second.
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 instruments in the noisy PC environment
SDA	Serial Data Analyzer

SDK	software development kit
settling time	the amount of time required for a voltage to reach its final value within specified limits
shared memory	*See* dual-access memory
signal conditioning	the manipulation of signals to prepare them for digitizing
signal divider	performing frequency division on an external signal
SIMM	single in-line memory module
SMB	a type of miniature coaxial signal connector
SNR	signal-to-noise ratio—the ratio of the overall rms signal level to the rms noise level, expressed in decibels
software trigger	a programmed event that triggers an event such as data acquisition
software triggering	a method of triggering in which you simulate an analog trigger using software. Also called conditional retrieval.
source impedance	a parameter of signal sources that reflects current-driving ability of voltage sources (lower is better) and the voltage-driving ability of current sources (higher is better)
SOURCE input pin	an counter input pin where the counter counts the signal transitions
S/s	samples per second—used to express the rate at which an instrument samples an analog signal
statically configured device	a device whose logical address cannot be set through software; that is, it is not dynamically configurable
switchless device	devices that do not require dip switches or jumpers to configure resources on the devices—also called Plug and Play devices
synchronous	(1) hardware—a property of an event that is synchronized to a reference clock (2) software—a property of a function that begins an operation and returns only when the operation is complete

system RAM	RAM installed on a personal computer and used by the operating system, as contrasted with onboard RAM
system noise	a measure of the amount of noise seen by an analog circuit or an ADC when the analog inputs are grounded

T

TC	terminal count—the highest value of a counter
throughput rate	the data, measured in bytes/s, for a given continuous operation, calculated to include software overhead. Throughput Rate = Transfer RateSoftware Overhead Factor.
top-level VI	VI at the top of the VI hierarchy. This term is used to distinguish the VI from its subVIs.
transducer	*See* sensor
transducer excitation	a type of signal conditioning that uses external voltages and currents to excite the circuitry of a signal conditioning system into measuring physical phenomena
transfer rate	the rate, measured in bytes/s, at which data is moved from source to destination after software initialization and set up operations; the maximum rate at which the hardware can operate
trigger	any event that causes or starts some form of data capture
TTL	transistor-transistor logic

U

UART	universal asynchronous receiver/transmitter—an integrated circuit that converts parallel data to serial data (and vice versa), commonly used as a computer bus to serial device interface for serial communication
UI	update interval
unipolar	a signal range that is always positive (for example, 0 to +10 V)

update	the output equivalent of a scan. One or more analog or digital output samples. Typically, the number of output samples in an update is equal to the number of channels in the output group. For example, one pulse from the update clock produces one update that sends one new sample to every analog output channel in the group.
update rate	the number of output updates per second

V

V	volts
V_{DC}	volts direct current
VDMAD	virtual DMA driver
VI	virtual instrument—(1) a combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument (2) a LabVIEW software module (VI), which consists of a front panel user interface and a block diagram program
V_{IH}	volts, input high
V_{IL}	volts, input low
V_{in}	volts in
VISA	virtual instrument software architecture—a new driver software architecture developed by National Instruments to unify instrumentation software GPIB, DAQ, and VXI. It has been accepted as a standard for VXI by the VXIplug&play Systems Alliance.
visual basic custom control (VBXs)	a specific form of binary packaged object that can be created by different companies and integrated into applications written using Visual Basic
V_{OH}	volts, output high
V_{OL}	volts, output low
VPICD	virtual programmable interrupt controller device
V_{ref}	reference voltage

W

waveform	multiple voltage readings taken at a specific sampling rate
wire	data path between nodes
word	the standard number of bits that a processor or memory manipulates at one time. Microprocessors typically use 8, 16, or 32-bit words.
working voltage	the highest voltage that should be applied to a product in normal use, normally well under the breakdown voltage for safety margin. See also Breakdown Voltage.

Index

A

application development, 2-1 to 2-14. *See also* board functions; channel functions.
- building LabVIEW applications, 2-3 to 2-4
- compiler information, 2-2
- dynamic link libraries, 2-1
- environments for application development, 1-1
- function overview, 2-4 to 2-8
- header files, 2-2
- import libraries
  - defined, 2-1
  - Microsoft Visual C++ and Borland C++ (table), 2-2
  - NISDA.LIB, 2-2
- overview, 2-1
- programming guidelines, 2-2
- programming with SDA instrument driver, 2-8 to 2-14
  - C implementation, 2-9 to 2-12
  - example application, 2-8 to 2-9
  - G implementation, 2-12 to 2-14

attribute functions. *See also* configuration functions.
- function tree for LabWindows/CVI software (table), 1-6
- niSda_ChannelGetAttribute, 3-43 to 3-46
- niSda_ChannelSetAttribute, 3-40 to 3-42

attributes
- list of SDA instrument driver software attributes (table), A-1 to A-3
- value attributes (table), B-1 to B-4

B

board functions
- configuration functions, 3-7 to 3-17
  - application development overview, 2-5
  - function tree for LabWindows/CVI software (table), 1-5
  - niSda_GetAttribute, 3-11 to 3-13
  - niSda_RecallSetup, 3-16 to 3-17
  - niSda_SaveSetup, 3-14 to 3-15
  - niSda_SetAttribute, 3-8 to 3-10
- life-cycle functions, 3-1 to 3-6
  - application development overview, 2-4
  - function tree for LabWindows/CVI software (table), 1-5
  - niSda_close, 3-5 to 3-6
  - niSda_init, 3-2 to 3-4
- list of functions, 3-1
- overview, 3-1
- utility functions, 3-18 to 3-28
  - application development overview, 2-5
  - function tree for LabWindows/CVI software (table), 1-5
  - niSda_error_message, 3-25 to 3-26
  - niSda_error_query, 3-23 to 3-24
  - niSda_reset, 3-19 to 3-20
  - niSda_revision_query, 3-27 to 3-28
  - niSda_self_test, 3-21 to 3-22

Borland C++ import library (table), 2-2

bulletin board support, D-1

C

C languages
implementation of SDA application example, 2-9 to 2-12
Microsoft Visual C++ and Borland C++ import libraries (table), 2-2
channel functions
application development overview, 2-6 to 2-8
attribute functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelGetAttribute, 3-43 to 3-46
niSda_ChannelSetAttribute, 3-40 to 3-42
configuration functions
function tree for LabWindows/CVI software (table), 1-5 to 1-6
niSda_ChannelConfigureGP, 3-32 to 3-33
niSda_ChannelConfigureRS232, 3-34 to 3-36
niSda_ChannelConfigureRS485, 3-37 to 3-39
niSda_ChannelReset, 3-30 to 3-31
function tree for LabWindows/CVI software (table), 1-5
list of functions, 3-29
overview, 3-29
transceiver functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelDisableClock Transceiver, 3-53 to 3-54
niSda_ChannelDisableData Tranceiver, 3-49 to 3-50
niSda_ChannelEnableClock Transceiver, 3-51 to 3-52
niSda_ChannelEnableData Tranceiver, 3-47 to 3-48
transmission functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelAbort, 3-77 to 3-78
niSda_ChannelCheckReception, 3-61 to 3-63
niSda_ChannelCheckTransmission, 3-73 to 3-74
niSda_ChannelInsertError, 3-75 to 3-76
niSda_ChannelReceive, 3-55 to 3-57
niSda_ChannelTransmit, 3-64 to 3-66
niSda_ChannelTransmitFile, 3-70 to 3-72
niSda_ChannelTransmitPattern, 3-67 to 3-69
trigger functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelTrigger, 3-79 to 3-80
compiler information, 2-2
configuration functions
board functions, 3-7 to 3-17
application development overview, 2-5
function tree for LabWindows/CVI software (table), 1-5
niSda_GetAttribute, 3-11 to 3-13
niSda_RecallSetup, 3-16 to 3-17
niSda_SaveSetup, 3-14 to 3-15
niSda_SetAttribute, 3-8 to 3-10
channel functions
function tree for LabWindows/CVI software (table), 1-5 to 1-6
niSda_ChannelConfigureGP, 3-32 to 3-33
niSda_ChannelConfigureRS232, 3-34 to 3-36
niSda_ChannelConfigureRS485, 3-37 to 3-39

niSda_ChannelReset, 3-30 to 3-31
customer communication, *xii,* D-1 to D-2

D

data types
compatible types and arrays (table), 1-2 to 1-3
overview, 1-2
documentation
conventions used in manual, *x*
National Instruments documentation, *xi*
organization of manual, *ix-x*
related documentation, *xi-12*

E

e-mail support, D-2
electronic support services, D-1 to D-2

F

fax and telephone support numbers, D-2
Fax-on-Demand support, D-2
FTP support, D-1
functions. *See* board functions; channel functions.

G

G language implementation of SDA application example, 2-12 to 2-14
channel configuration (figure), 2-13
channel transmission and reception operation (figure), 2-14
closing (figure), 2-14
initialization and board configuration (figure), 2-12

H

header files
including driver function prototypes (note), 1-3
NISDA.H, 2-2

I

import libraries
defined, 2-1
Microsoft Visual C++ and Borland C++ (table), 2-2
NISDA.LIB, 2-2

L

LabVIEW software
building applications, 2-3 to 2-4
overview, 1-8
SDA VIs available (figure), 1-4
LabWindows/CVI software
function tree for SDA (table), 1-5 to 1-6
overview, 1-8 to 1-9
life-cycle functions, 3-1 to 3-6
application development overview, 2-4
function tree for LabWindows/CVI software (table), 1-5
niSda_close, 3-5 to 3-6
niSda_init, 3-2 to 3-4

M

manual. *See* documentation.
Microsoft Visual C++ import library (table), 2-2

N

niSda_ChannelAbort function, 3-77 to 3-78
niSda_ChannelCheckReception function, 3-61 to 3-63
niSda_ChannelCheckTransmission function, 3-73 to 3-74
niSda_ChannelConfigureGP function, 3-32 to 3-33
niSda_ChannelConfigureRS232 function, 3-34 to 3-36
niSda_ChannelConfigureRS485 function, 3-37 to 3-39
niSda_ChannelDisableClockTransceiver function, 3-53 to 3-54
niSda_ChannelDisableDataTranceiver function, 3-49 to 3-50
niSda_ChannelEnableClockTransceiver function, 3-51 to 3-52
niSda_ChannelEnableDataTranceiver function, 3-47 to 3-48
niSda_ChannelGetAttribute function, 3-43 to 3-46
niSda_ChannelInsertError function, 3-75 to 3-76
niSda_ChannelReceive function, 3-55 to 3-57
niSda_ChannelReset function, 3-30 to 3-31
niSda_ChannelSetAttribute function, 3-40 to 3-42
niSda_ChannelTransmit function, 3-64 to 3-66
niSda_ChannelTransmitFile function, 3-70 to 3-72
niSda_ChannelTransmitPattern function, 3-67 to 3-69
niSda_ChannelTrigger function, 3-79 to 3-80
niSda_close function, 3-5 to 3-6
niSda_error_message function, 3-25 to 3-26
niSda_error_query function, 3-23 to 3-24
niSda_GetAttribute function, 3-11 to 3-13
NISDA.H header file, 2-2
niSda_init function, 3-2 to 3-4
niSda_RecallSetup function, 3-16 to 3-17
niSda_reset function, 3-19 to 3-20
niSda_revision_query function, 3-27 to 3-28
niSda_SaveSetup function, 3-14 to 3-15
niSda_self_test function, 3-21 to 3-22
niSda_SetAttribute function, 3-8 to 3-10

P

programming language considerations, 1-3 to 1-7
  application development environments, 1-1
  code examples, 1-7
  LabVIEW environment, 1-4
  LabWindows/CVI environment, 1-4 to 1-6
programming with SDA instrument driver. *See* application development.
prototypes, including (note), 1-3

S

SDA instrument driver software. *See also* board functions; channel functions.
  application development environments, 1-1
  architecture (figure), 1-7
  attributes (table), A-1 to A-3
  data types
    compatible types and arrays (table), 1-2 to 1-3
    overview, 1-2
  interface, 2-1
  loading from CD-ROM, 1-1
  overview, 1-9
  programming language considerations, 1-3 to 1-7
    code examples, 1-7
    LabVIEW environment, 1-4

LabWindows/CVI environment, 1-4 to 1-6
requirements for getting started, 1-8
software programming choices, 1-8 to 1-9
driver software, 1-9
National Instruments application software, 1-8 to 1-9
status codes (table), C-1 to C-3
value attributes (table), B-1 to B-4
Serial Data Analyzer instrument driver. *See* SDA instrument driver software.
software programming choices, 1-8 to 1-9. *See also* programming language considerations.
driver software, 1-9
National Instruments application software, 1-8 to 1-9
status codes
format, 1-2
list of codes (table), C-1 to C-3

T

technical support, D-1 to D-2
telephone and fax support numbers, D-2
transceiver functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelDisableClockTransceiver, 3-53 to 3-54
niSda_ChannelDisableDataTranceiver, 3-49 to 3-50
niSda_ChannelEnableClockTransceiver, 3-51 to 3-52
niSda_ChannelEnableDataTranceiver, 3-47 to 3-48
transmission functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelAbort, 3-77 to 3-78
niSda_ChannelCheckReception, 3-61 to 3-63
niSda_ChannelCheckTransmission, 3-73 to 3-74
niSda_ChannelInsertError, 3-75 to 3-76
niSda_ChannelReceive, 3-55 to 3-57
niSda_ChannelTransmit, 3-64 to 3-66
niSda_ChannelTransmitFile, 3-70 to 3-72
niSda_ChannelTransmitPattern, 3-67 to 3-69
trigger functions
function tree for LabWindows/CVI software (table), 1-6
niSda_ChannelTrigger, 3-79 to 3-80

U

utility functions, 3-18 to 3-28
application development overview, 2-5
function tree for LabWindows/CVI software (table), 1-5
niSda_error_message, 3-25 to 3-26
niSda_error_query, 3-23 to 3-24
niSda_reset, 3-19 to 3-20
niSda_revision_query, 3-27 to 3-28
niSda_self_test, 3-21 to 3-22

V

value attributes (table), B-1 to B-4
variable data types (table), 1-2 to 1-3

W

Windows application development. *See* application development.