I²C/SPI

NI-845*x* Hardware and Software Manual

June 2011
371746C-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

`ni.com`

Worldwide Offices

Visit `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

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

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at `ni.com/info` and enter the Info Code `feedback`.

Important Information

Warranty

The NI USB-845*x* hardware 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 document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

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

Copyright

Trademarks

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

Export Compliance Information

Refer to the *Export Compliance Information* at `ni.com/legal/export-compliance` for the National Instruments global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

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

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

Compliance

Electromagnetic Compatibility Information

This hardware has been tested and found to comply with the applicable regulatory requirements and limits for electromagnetic compatibility (EMC) as indicated in the hardware's Declaration of Conformity (DoC)[1]. These requirements and limits are designed to provide reasonable protection against harmful interference when the hardware is operated in the intended electromagnetic environment. In special cases, for example when either highly sensitive or noisy hardware is being used in close proximity, additional mitigation measures may have to be employed to minimize the potential for electromagnetic interference.

While this hardware is compliant with the applicable regulatory EMC requirements, there is no guarantee that interference will not occur in a particular installation. To minimize the potential for the hardware to cause interference to radio and television reception or to experience unacceptable performance degradation, install and use this hardware in strict accordance with the instructions in the hardware documentation and the DoC[1].

If this hardware does cause interference with licensed radio communications services or other nearby electronics, which can be determined by turning the hardware off and on, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the antenna of the receiver (the device suffering interference).
- Relocate the transmitter (the device generating interference) with respect to the receiver.
- Plug the transmitter into a different outlet so that the transmitter and the receiver are on different branch circuits.

Some hardware may require the use of a metal, shielded enclosure (windowless version) to meet the EMC requirements for special EMC environments such as, for marine use or in heavy industrial areas. Refer to the hardware's user documentation and the DoC[1] for product installation requirements.

When the hardware is connected to a test object or to test leads, the system may become more sensitive to disturbances or may cause interference in the local electromagnetic environment.

Operation of this hardware in a residential area is likely to cause harmful interference. Users are required to correct the interference at their own expense or cease operation of the hardware.

Changes or modifications not expressly approved by National Instruments could void the user's right to operate the hardware under the local regulatory rules.

[1] The Declaration of Conformity (DoC) contains important EMC compliance information and instructions for the user or installer. To obtain the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the Certification column.

Contents

About This Manual

Chapter 1
Introduction

Chapter 2
Installation

Chapter 3
NI USB-845*x* Hardware Overview

Chapter 4
Using the NI-845*x* API

Chapter 5
Using the NI-845*x* I^{2}C API

Chapter 6
NI-845*x* I^2C API for LabVIEW

Chapter 7
NI-845*x* I²C API for C

Chapter 8
Using the NI-845*x* SPI API

Chapter 9
NI-845*x* SPI API for LabVIEW

Chapter 10
NI-845*x* SPI API for C

Chapter 11
Using the NI-845*x* SPI Stream API

Chapter 12
NI-845*x* SPI Stream API for LabVIEW

Chapter 13
NI-845*x* SPI Stream API for C

Chapter 14
Using the NI-845*x* DIO API

Chapter 15
NI-845*x* DIO API for LabVIEW

Chapter 16
NI-845*x* DIO API for C

About This Manual

This manual explains how to use the NI-845*x* software. It contains installation and configuration information, function reference for a LabVIEW or C-based API, and a USB-845*x* hardware overview and specifications.

Use this manual to learn the basics of I^{2}C and SPI communication with NI-845*x*, as well as how to develop an application.

Conventions

The following conventions are used in this manual:

» — The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **Options»Settings»General** directs you to pull down the **Options** menu, select the **Settings** item, and select **General** from the last dialog box.

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

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash. When this symbol is marked on a product, refer to the *Safety* section in Appendix A, *NI USB-845x Hardware Specifications*, for information about precautions to take.

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

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

`monospace` — Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

1 Introduction

This chapter introduces the Inter-IC (I^2C) and Serial Peripheral Interface (SPI) buses.

I^2C Bus

NXP (formerly Philips Semiconductors) developed the I^2C bus in the early 1980s to connect a CPU to peripheral chips in televisions. I^2C is also used to communicate with temperature sensors, EEPROMs, LCD displays, and other embedded peripheral devices.

I^2C Terminology

This manual uses the following I^2C bus terms:

I^2C	Inter-IC.
SMBus	System Management Bus.
Transmitter	Device transmitting data on the bus.
Receiver	Device receiving data from the bus.
Master	Device that can initiate and terminate a transfer on the bus. The master is responsible for generating the clock (SCL) signal.
Master Code	Unique 3-bit code designated to each High Speed master to identify the master initiating a High Speed operation and arbitrate the I^2C bus.
Slave	Device addressed by the master.
Multimaster	The ability for more than one master to co-exist on the bus concurrently without data loss.

Arbitration	The procedure to allow multiple masters to determine which single master controls the bus for a particular transfer time.
Synchronization	The defined procedure to allow the clock signals provided by two or more masters to be synchronized.
SDA	Serial DAta (data signal line).
SCL	Serial CLock (clock signal line).

I^2C Bus

The I^2C bus is a two-wire half-duplex serial interface. The two wires, SDA and SCL, are both bidirectional. The I^2C specification version 3.0 defines four speed categories: Standard mode at up to 100 kbits/s, Fast mode at up to 400 kbits/s, Fast mode Plus at up to 1 Mbits/s, and High Speed mode at up to 3.4 Mbits/s.

Each device connected to the I^2C bus has a unique 7-bit I^2C address to facilitate identification and communication by the master. Typically, the upper four bits are fixed and assigned to specific categories of devices (for example, 1010 is assigned to serial EEPROMs). The three lower bits are programmable through hardware address pins, allowing up to eight devices of the same type to be connected to a single I^2C bus.

Each device on the bus (both master and slave) can be a receiver and/or transmitter. For example, an LCD is typically only a receiver, while an EEPROM is both a transmitter and receiver.

The I^2C is a multimaster bus, meaning that multiple masters can be connected to the bus at the same time. While a master is initiating a transfer on the bus, all other devices, including other masters, are acting like slaves. However, if another master is trying to control the bus at the same time, I^2C defines an arbitration mechanism to determine which master gets control of the bus.

I^2C Arbitration

When two masters are trying to control the bus simultaneously, or if a second master joins the bus in the middle of a transfer and wants to control the bus, the I^2C bus has an arbitration scheme to guarantee no data corruption.

With I^2C, a line (both SDA and SCL) is either driven low or allowed to be pulled high. When a master changes a line state to high, it must sample the line afterwards to make sure it really has been pulled high. If the master samples the SDA bus after setting it high, and the sample shows that the line is low, it knows another master is driving it low. The master assumes it has lost arbitration and waits until it detects a stop condition before making another attempt to start transmitting.

When in High Speed mode, arbitration occurs only during the master code transfer. Each master code must be unique on the I^2C bus so the arbitration is finalized once the entire master code has been transferred.

I^2C Transfers

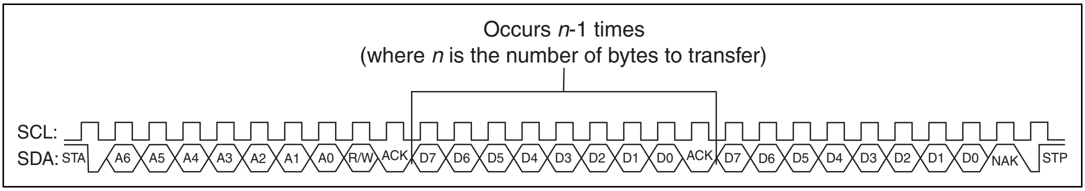


Figure 1-1. I^2C Transfers

To initiate a transfer, the master issues a start condition by changing the SDA line level from high to low while keeping the SCL clock line high. When this occurs, the bus is considered busy, and all devices on the bus get ready to listen for incoming data.

Next, the master sends the 7-bit address and 1-bit data transfer direction on the bus to configure for the appropriate data transfer. All slaves compare the address with their own address. If the address matches, the slave produces an acknowledge signal.

If the master detects an acknowledge signal, it starts transmitting or receiving data. To transmit data to a device, the master places the first bit onto the SDA line and generates a clock pulse to transmit the bit across the bus to the slave. To receive data from a device, the master releases the SDA line, allowing the slave to take control of it. The master generates a clock pulse on the SCL line for each bit, reading the data while the SCL line is high. The device is not allowed to change the SDA line state while the SCL line is high.

After the data transmission, the master issues the stop condition by changing the SDA line from low to high while keeping the SCL clock line high. When this occurs, the bus is considered free again for another master to initiate a data transfer.

For High Speed mode, the transfer is initiated with a start condition followed by a master code transmitted at a non-High Speed clock rate. Because master codes are unique on the I^2C bus, the master code never should be followed by an acknowledge signal. Once the master code has been transmitted, a restart condition is transmitted followed by the control byte and data transmitted at a High Speed clock rate.

I^2C Clock Stretching

Because the master controls the clock, the I^2C specification provides a mechanism to allow the slave to slow down the bus traffic when it is not ready. This mechanism is known as clock stretching. When not in High Speed mode, a slave may additionally hold down SCL to prevent it from rising high again to slow down the SCL clock rate or pause I^2C communication during any SCL low phase. When in High Speed mode, SCL may be stretched only after the reception and acknowledgement of a byte.

When the master attempts to make SCL high to complete the current clock pulse, it must verify that it has really gone high. If it is still low, it knows a slave is holding it low and must wait until it goes high before continuing.

I^2C Extended (10-Bit) Addressing

Typical I^2C devices use a 7-bit addressing scheme. I^2C also defines a 10-bit addressing scheme that allows up to 1024 additional addresses to be connected to the I^2C bus. This 10-bit addressing scheme does not affect the existing 7-bit addressing, allowing both 7-bit and 10-bit addressed devices to share the bus. A device that supports 10-bit addressing receives the address across two bytes. The first byte consists of the NXP-designated 10-bit slave address group (11110), the 2 MSBs of the device address, and the Read/Write bit. The next data byte sent across the bus contains the eight LSBs of the address.

I^2C High Speed Master Code

For High Speed mode, the NXP specification defines a master code transferred in Standard, Fast, or Fast mode Plus to arbitrate the I^2C bus. All High Speed masters must have a master code defined, and all master codes must be unique on the bus. The master code consists of the NXP-designated master code address group (00001), then the three master code bits. This allows up to eight High Speed masters to be connected to the High Speed I^2C bus; however, the NXP I^2C specification describes master code 0 as reserved for test and diagnostic purposes.

I^2C vs. SMBus

Intel defined the System Management Bus (SMBus) in 1995. This bus is used primarily in personal computers and servers for low-speed system management communications.

The I^2C bus and SMBus are very similar; at frequencies at or below 100 kHz, they tend to be interchangeable. However, the following sections describe some important differences.

Timeout and Clock Rates

I^2C has no minimum clock rate, and as such there is no minimum clock frequency duration. However, SMBus does not allow the clock to be slower than 10 kHz; a device will reset if the clock remains low for more than 35 ms.

I^2C allows clock rates of 100 kHz, 400 kHz, 1 MHz, and 3.4 MHz, whereas SMBus is limited to a maximum clock rate of 100 kHz.

Logic Levels

Logic high is defined on I^2C as 0.7 * V_{DD}. On SMBus, logic high is defined as 2.1 V.

Logic low is defined on I^2C as 0.3 * V_{DD}. On SMBus, logic low is defined as 0.8 V.

Current Levels

The sink current also varies between I^2C and SMBus. In I^2C, the maximum is 3 mA for Standard and High Speed mode. For Fast mode, the maximum sink current is 6 mA, and Fast mode Plus allows 20 mA. SMBus has a maximum of 350 µA. This determines the lowest acceptable value of the pull-up resistor. At 3 V in Standard mode, an I^2C bus should have a pull-up of > 1 kΩ; SMBus should have a pull-up of > 8.5 kΩ. However, many SMBus systems violate this rule; a common range for both SMBus and I^2C tends to be in the 2.4–3.9 kΩ range, but may vary significantly for various speeds and bus capacitance ranges.

For more information about I^2C current limitations and pullup resistor selection, refer to the NXP I^2C specification.

Throughout this document, we will refer to the bus as an I^2C bus. For information about compatibility of your NI 845*x* device with SMBus, refer to Chapter 3, *NI USB-845x Hardware Overview*.

SPI Bus

The SPI bus is a de facto standard originated by Motorola and is used to communicate with devices such as EEPROMs, real-time clocks, converters (ADC and DAC), and sensors. Implementations may vary, as SPI does not have a formal specification.

SPI Terminology

This manual uses the following SPI bus terms:

CLK	CLocK. The clock is generated by the master device and controls when data is sent and read.
MOSI	Master Output, Slave Input. The MOSI line carries data from the master to the slave.
MISO	Master Input, Slave Output. The MISO carries data from the slave to the master.
CS or SS	Chip Select or Slave Select. Connection from the master to a slave that signals the slave to listen for SPI clock and data signals.
CPOL	Clock POLarity. The polarity indicating whether the clock makes positive or negative pulses.
CPHA	Clock PHAse. This controls the positioning of the data bits relative to the clock edges.
Shift Register	A shift register is connected to the MOSI and MISO lines. As data is read from the input, it is placed into the shift register. Data from the shift register is placed into the output, creating a full-duplex communication loop.
Master	The master device provides the clock signal and determines the chip select line state.

Slave	The slave device receives the clock and chip select from the master. The maximum number of slaves is dependent on the number of available chip select lines.

SPI Bus

The SPI bus is a four-wire, full-duplex serial interface. Three of the wires, SCK, MOSI, and MISO, are shared along with a fourth wire, known as the chip select, which is a direction connection between the master and a single slave.

Communication across SPI uses a system known as data exchange. Whenever a bit is written to an SPI device across the MOSI lines, the SPI device concurrently returns a bit on the MISO line. Because data is transferred in both directions, it is up to the receiving device to know whether the received by is meaningful or not. For example, to receive data from an EEPROM, the master must configure the EEPROM to send *n* bytes of data and then must send *n* bytes to be exchanged for valid data. These bytes can usually be any value, and writing them serves only to clock the data out of the receiving device.

Clock and Polarity

Parameters called clock polarity (CPOL) and clock phase (CPHA) determine the clock idle state and the edge of the clock signal when the data is driven and sampled. These parameters are sometimes expressed as four modes, as shown in Table 1-1.

Table 1-1. SPI Modes

SPI Mode	Polarity	Phase
0	0	0
1	0	1
2	1	0
3	1	1

When the polarity is 0, the clock idles low. When the polarity is 1, the clock idles high. When the phase is 0, data is latched at the clock transition from idle to asserted. When the phase is 1, the data is latched at the clock transition from asserted to idle. Figure 1-2 shows how the four SPI modes affect the clock and sample times.

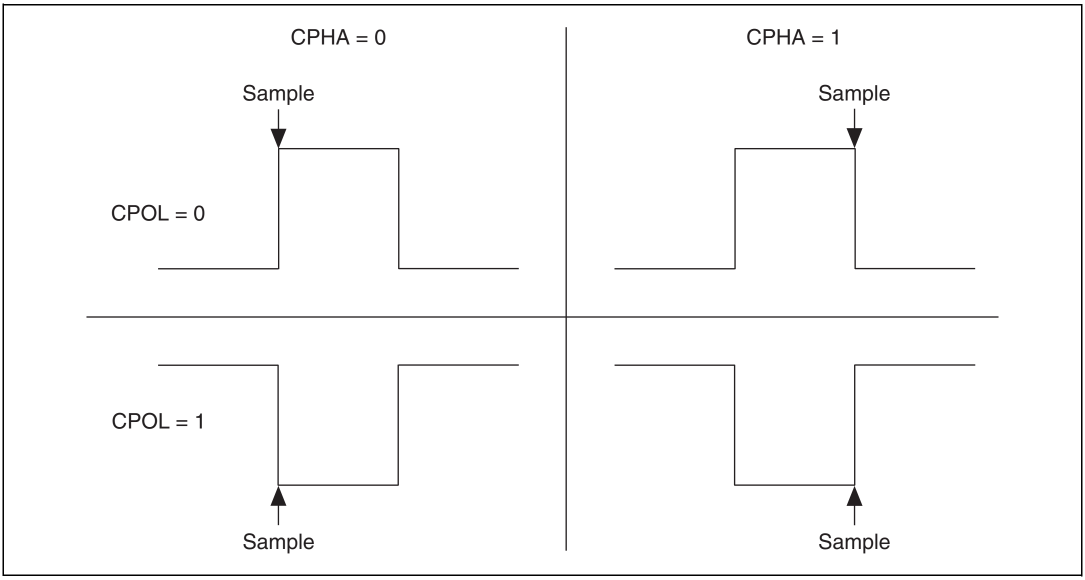


Figure 1-2. SPI Polarity Phase Differences

Error Handling

Unlike I^2C, SPI has no acknowledgement mechanism or flow control. This prevents the SPI master from knowing whether a slave received a data byte correctly or even whether it is connected to the bus.

2

Installation

This chapter explains how to install the NI-845*x* software and hardware.

Software Installation

This section discusses installing the NI-845*x* software on Microsoft Windows.

Note You need administrator privileges to install the NI-845*x* software on your computer.

1. Insert the *NI-845x Software* CD into your CD-ROM drive. The installer launches if your CD-ROM drive plays data CDs automatically. If the installer does not launch automatically, navigate to the CD using Windows Explorer and launch the `autorun.exe` file from your *NI-845x Software* CD.
2. The Installation Wizard guides you through the necessary steps to install the NI-845*x* software. You can go back and change values where appropriate by clicking the **Back** button. You can exit the setup where appropriate by clicking **Cancel**.
3. When installation is complete, select **Finish**.

Hardware Installation

Step 1: Unpack the Devices, Accessories, and Cables

Your device ships in an antistatic package to prevent electrostatic discharge (ESD) damage to the device. ESD can damage several components on the device.

To avoid such damage, take the following precautions:

- Ground yourself using a grounding strap or by touching a grounded object.
- Touch the antistatic package to a metal part of the computer chassis before removing the device from the package.

Remove the device from the package and inspect the device for loose components or any sign of damage. Notify National Instruments if the device appears damaged in any way. Do not install a damaged device into your computer or PXI chassis.

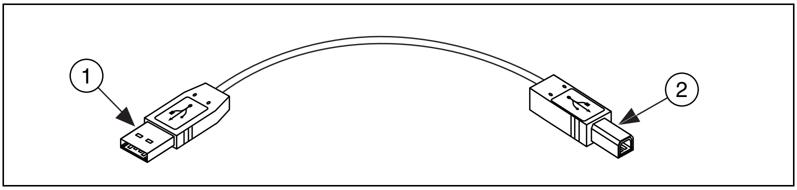
Store the device in the antistatic package when the device is not in use.

For safety and compliance information, refer to the device documentation packaged with your device.

Step 2: Install the Devices, Accessories, and Cables

Complete the following steps to install an NI USB device:

1. Connect the USB cable from the computer USB port or from any other hub that provides USB power to the USB port on the device. The following figure shows the USB cable and its connectors.



1 Host/Hub/PC USB Port	2 NI USB Device USB Port

2. Power on your computer or PXI chassis. On some Windows systems, the Found New Hardware wizard opens with a dialog box for every device installed. Click **Next** or **Yes** to install the software for each device.
3. Install accessories and/or terminal blocks according to the instructions in their user guides.

Step 3: Confirm that Your Device Is Recognized

To verify that the USB device is recognized, complete the following steps:

1. Double-click the **Measurement & Automation** icon on the desktop to open Measurement & Automation Explorer (MAX).
2. Expand **Devices and Interfaces**.
3. Verify that the device appears under **USB Devices**. If the device does not appear, press <F5> to refresh the view in MAX. If the device is still not recognized, refer to `ni.com.support/install` for troubleshooting information.

3

NI USB-845*x* Hardware Overview

Overview

NI USB-845*x* modules are USB 2.0 devices that provide I^2C and SPI connectivity along with general-purpose DIO lines.

NI USB-8451

Overview

The NI USB-8451 is a full-speed USB 2.0 device that provides I^2C (up to 250 KHz) and SPI (up to 12 MHz) connectivity, along with eight SPI chip select lines and eight general-purpose DIO lines.

The NI USB-8451 is available in an enclosure and as a board-only version. In this manual, the enclosure version is referred to as the NI USB-8451, and the board-only version is referred to as the NI USB-8451 OEM. Unless otherwise noted, all information in this manual applies to both the NI USB-8451 and NI USB-8451 OEM.

Block Diagram

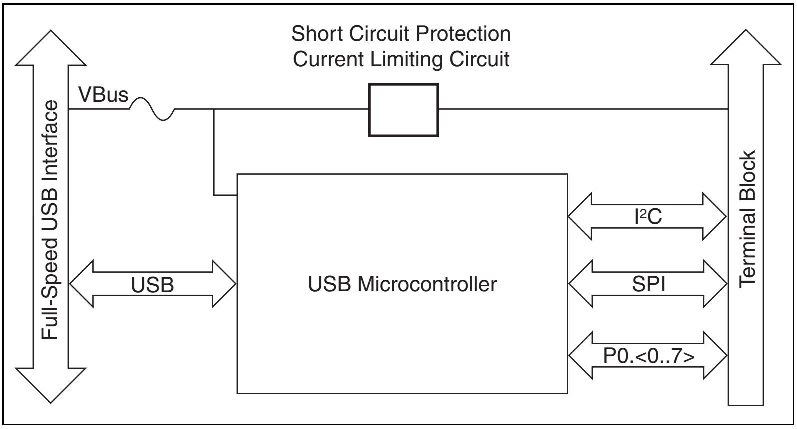


Figure 3-1. NI USB-8451 Block Diagram

Installing Software

Install the software provided with the NI USB-8451 or NI USB-8451 OEM module. Refer to the *NI-845x Software and Hardware Installation Guide* for more information.

Setting Up Hardware

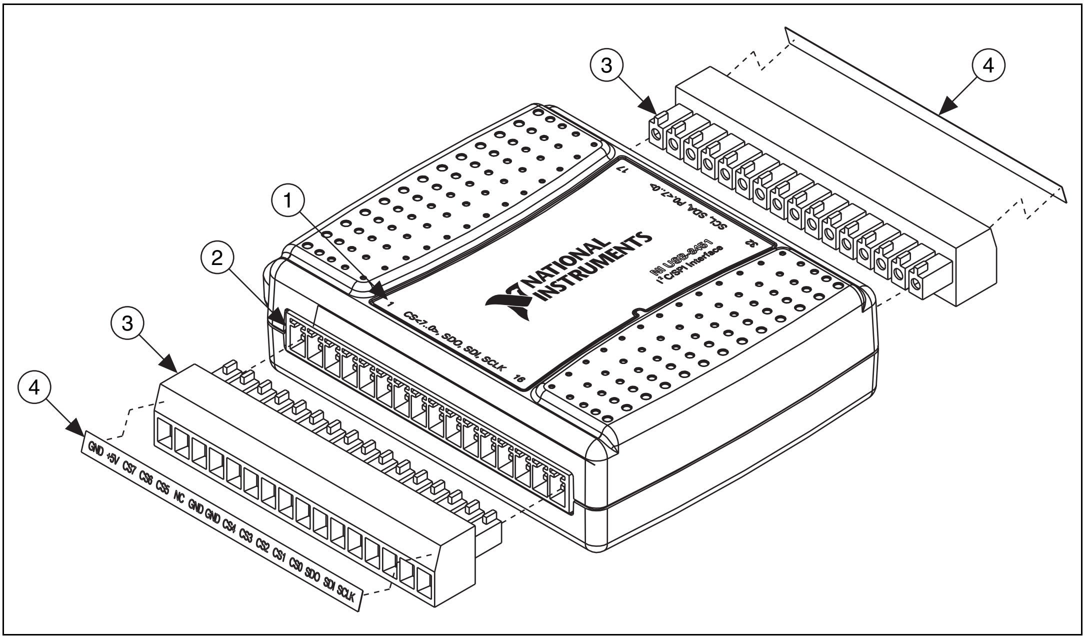
NI USB-8451

Complete the following steps to set up the hardware:

1. Install the combicon screw terminal blocks by inserting them into the combicon jacks.

Note The NI USB-8451 kit ships with signal labels. You can apply the signal labels to the screw terminal blocks for easy signal identification.

2. Refer to Table 3-1 and Figure 3-2 for label orientation and affix the provided signal labels to the screw terminal blocks. Until the signal labels are applied, you can insert the screw terminal blocks into either combicon jack.



1	Overlay Label with Pin Orientation Guides	3	Screw Terminal Blocks
2	Combicon Jack	4	Signal Labels

Figure 3-2. Signal Label Application Diagram

Note Once you label the screw terminal blocks, you must insert them into only the matching combicon jacks, as the overlay label on the NI USB-8451 device indicates.

3. Connect the wiring to the appropriate screw terminals.

NI USB-8451 OEM

The NI USB-8451 OEM board has a USB Series B-type receptacle for connection to the host machine. For the front-end I/O, the board has a 34-pin IDC ribbon cable header. Use any 34-pin female IDC (ribbon) cable to access the I/O.

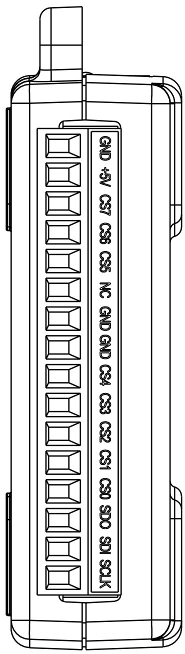
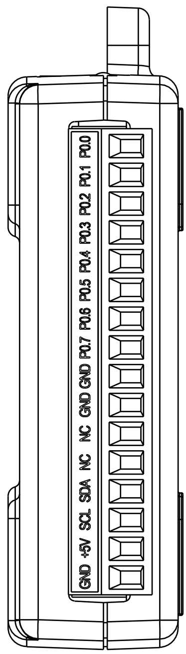
I/O Connector and Cable

NI USB-8451

The NI USB-8451 ships with two detachable terminal blocks for digital signals. The individual terminals accept 16 AWG to 28 AWG wire.

Table 3-1 lists the digital terminal assignments.

Table 3-1. Digital Terminal Assignments

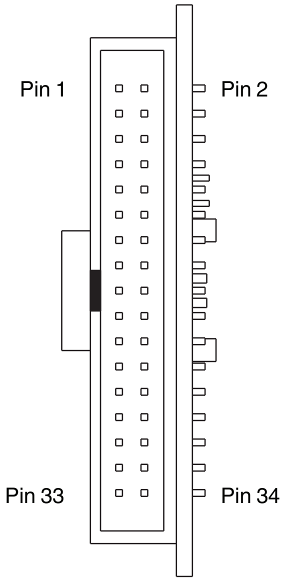
Module	Terminal	Signal	Module	Terminal	Signal
	1	GND		17	P0.0
	2	+5 V		18	P0.1
	3	SPI CS 7		19	P0.2
	4	SPI CS 6		20	P0.3
	5	SPI CS 5		21	P0.4
	6	NC		22	P0.5
	7	GND		23	P0.6
	8	GND		24	P0.7
	9	SPI CS 4		25	GND
	10	SPI CS 3		26	GND
	11	SPI CS 2		27	NC
	12	SPI CS 1		28	NC
	13	SPI CS 0		29	I^2C SDA
	14	SPI MOSI (SDO)		30	I^2C SCL
	15	SPI MISO (SDI)		31	+5 V
	16	SPI CLK (SCLK)		32	GND

NI USB-8451 OEM

Use any 34-pin female IDC (ribbon) cable to connect to the IDC connector on the NI USB-8451 OEM.

Table 3-2 lists the pin assignments and signal names for the IDC connector.

Table 3-2. Pin Assignments

Signal	Pin	Connector	Pin	Signal
NC	1		2	GND
NC	3		4	SCLK
SDA	5		6	GND
SCL	7		8	MISO
NC	9		10	GND
CS5	11		12	MOSI
CS6	13		14	GND
CS7	15		16	CS0
P0.0	17		18	GND
P0.1	19		20	CS1
P0.2	21		22	GND
P0.3	23		24	CS2
P0.4	25		26	GND
P0.5	27		28	CS3
P0.6	29		30	GND
P0.7	31		32	CS4
+5V	33		34	+5V

Signal Descriptions

Table 3-3 describes the signals available on the I/O connectors.

Table 3-3. Signal Descriptions

Signal Name	Direction	Description
SPI CS <0..7>	Output	**Chip Select Signals**—Outputs used to select the desired SPI peripheral device.
SPI MOSI (SDO)	Output	**Master Output Slave Input**—SPI communication signal to slave device.
SPI MISO (SDI)	Input	**Master Input Slave Output**—SPI communication signal from slave device.
SPI CLK (SCLK)	Output	**SPI Clock**—SPI output clock signal to slave devices capable of clock rates up to 12 MHz.
I^{2}C SDA	Open-drain	**I^{2}C Serial Data**—Data signal for I^{2}C communication.
I^{2}C SCL	Open-drain	**I^{2}C Clock**—I^{2}C clock signal to slave devices capable of clock rates up to 250 kHz.
P0.<0..7>	Input or output	**Digital I/O Signals**—You can individually configure each signal as an input or output. You can configure the port for open-drain or push-pull output.[1]
+5 V	Output	**+5 V**—The voltage source provided by the USB host. The voltage is nominally 5 V, but varies from system to system.
GND	—	**Ground**—The reference for the digital signals and the +5 VDC supply.
NC	—	**No Connect**—Do not connect any signals to this terminal.

[1] If you configure the DIO port for open-drain output, you must supply pull-up resistors to V_{cc} (3.3 or 5 V). The resistor value must not be lower than 1 kΩ.

Front-End I/O Interfaces

Digital I/O (DIO)

The NI USB-8451 (and NI USB-8451 OEM) has eight single-ended digital lines, P0.<0..7>.

You can program each DIO line individually as a static DI or DO line. You can use static DIO lines to monitor or control digital signals. All samples of static DI lines and updates of DO lines are software timed.

The default configuration of the DIO port is push-pull, allowing 3.3 V operations. To achieve 5 V operation, change the output driver type to open-drain and add an external pull-up resistor (Rp), as shown in Figure 3-3. Do not use a pull-up resistor of less than 1 kΩ.

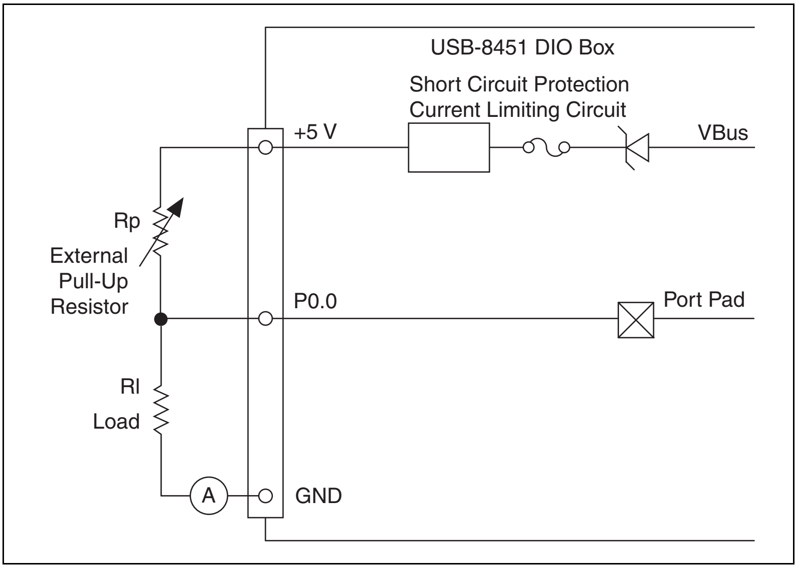
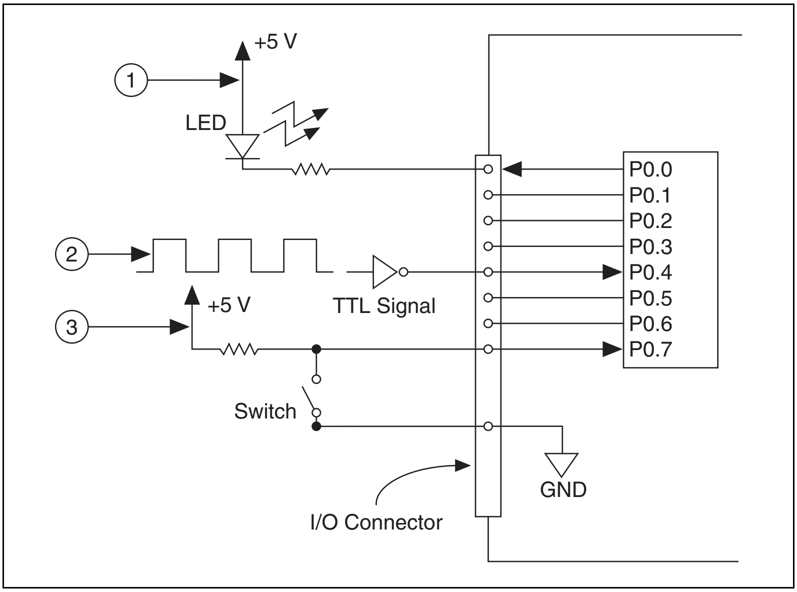


Figure 3-3. Example of Connecting External User-Provided Resistor

Figure 3-4 shows P0.<0..7> connected to example signals configured as digital inputs and digital outputs. Refer to Figure 3-4 for some common examples of connections of DIO lines with standard circuits.



1	P0.0 Configured as an Open-Drain Digital Output Driving an LED
2	P0.4 Configured as a Digital Input Receiving a TTL Signal from a Gated Invertor
3	P0.7 Configured as a Digital Input Receiving a 0 V or 5 V Signal from a Switch

Figure 3-4. Example of Connecting a Load

Caution Exceeding the maximum input voltage ratings or maximum output ratings, which are listed in Appendix A, *NI USB-845x Hardware Specifications*, can damage the USB device and the computer. National Instruments is not liable for any damage resulting from such signal connections.

SPI Interface

Figure 3-5 shows a typical SPI interface to three peripherals. All devices share the SPI MISO, SPI MOSI, and SPI CLK signals. Each peripheral has its own CS signal for addressing it.

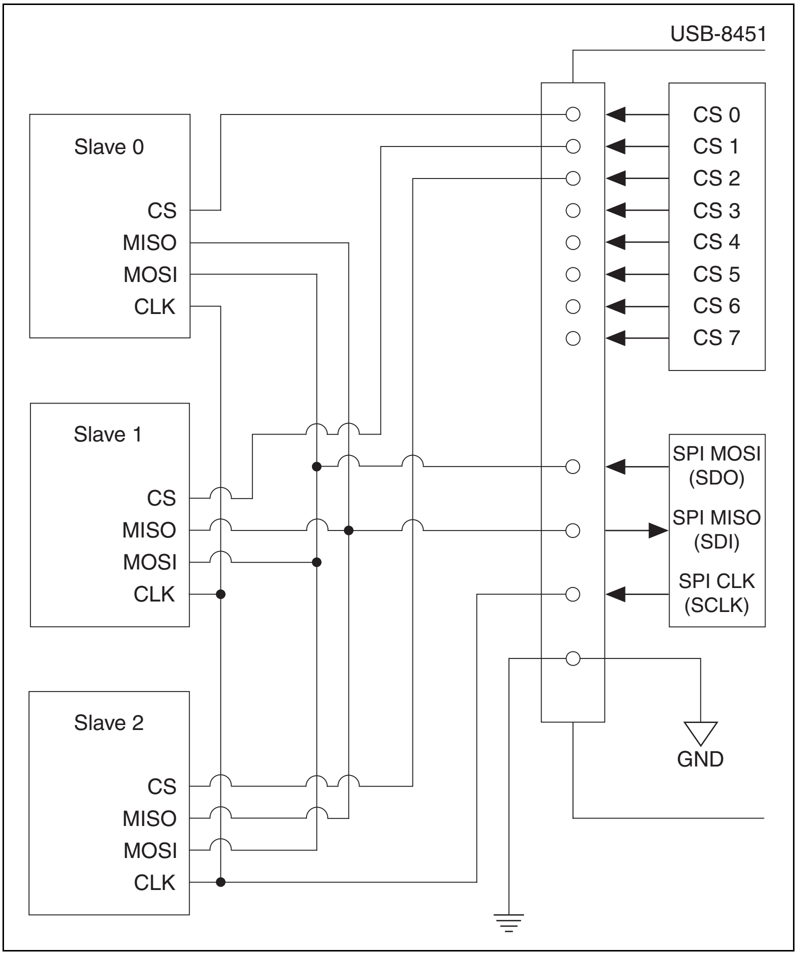


Figure 3-5. SPI Interface to Three Peripherals

I²C Interface

Figure 3-6 shows a typical I²C interface to two peripherals. All devices on the I²C bus share the SDA and SCL signals. SDA and SCL must be pulled up externally. Refer to the I²C specification to select the correct resistor values for your bus.

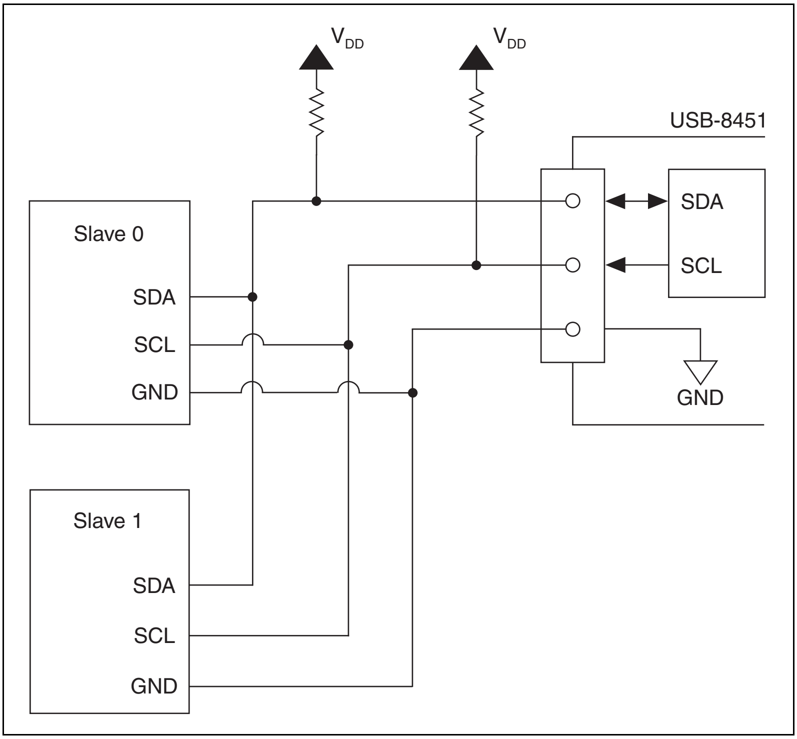


Figure 3-6. I²C Interface to Two Peripherals

I/O Protection

Each DIO, SPI, and SPI CS signal is protected against overvoltage, undervoltage, and overcurrent conditions, as well as ESD events. However, you should avoid these fault conditions by following these guidelines:

- If you configure a line as an output, do not connect it to any external signal source, ground signal, or power supply.
- If you configure a line as an output, understand the current requirements of the load connected to these signals. Do not exceed the specified current output limits of the module.

- If you configure a line as an input, do not drive the line with voltages outside its normal operating range.
- Treat the module as you would treat any static-sensitive device. Always properly ground yourself and the equipment when handling the USB device or connecting to it.

Caution Take special care with respect to the I^2C SDA and SCL lines. To allow for external pull-ups, the circuit protection has been removed. Do not exceed the specified voltages for these signals.

Power-On States

At system startup and reset, the hardware sets all DIO lines to high-impedance inputs. The module does not drive any of the signals high or low.

+5 V Power Source

The NI USB-8451 (and NI USB-8451 OEM) supplies a nominal 5 V from two pins, one on each screw terminal block. The USB host provides the voltage source. The voltage is nominally 5 V, but varies from system to system. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for more information about USB bus power specifications. You can use this source to power external components.

Note While the device is in USB suspend, the output is disabled.

Caution When using the 5 V source, understand the current requirements of the load connected. Do not exceed the specified current USB Vbus output limits.

NI USB-8452

Overview

The NI USB-8452 is a high-speed USB device featuring both I^2C (up to 3.3 MHz) and SPI (up to 50 MHz) connectivity along with eight chip select lines and eight general-purpose DIO lines. The NI USB-8452 has a programmable reference voltage to allow communication using I^2C, SPI, and DIO at multiple logic levels.

The NI USB-8452 is available in a board-only packaging only. In this manual, it is referred to as the NI USB-8452 OEM.

Block Diagram

The block diagram in Figure 3-7 shows key NI USB-8452 OEM module functional components.

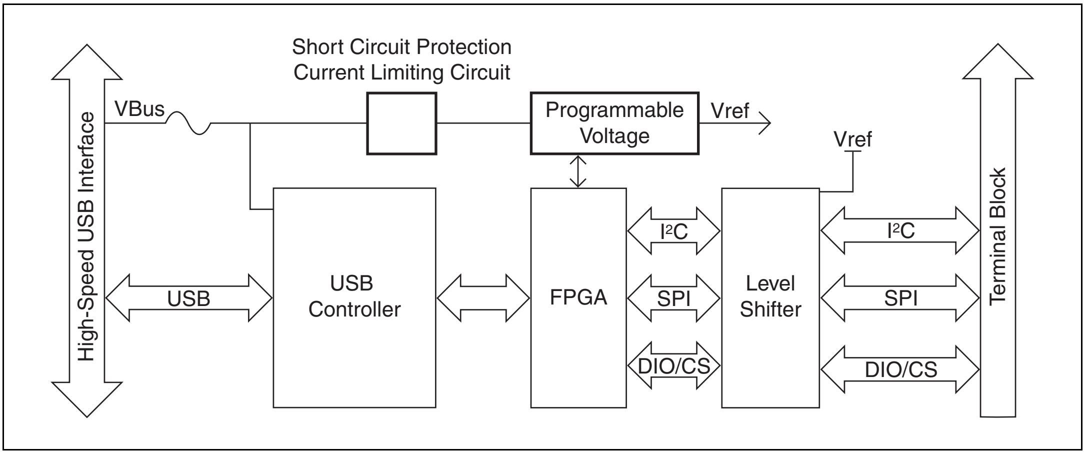


Figure 3-7. NI USB-8452 OEM Block Diagram

The NI USB-8452 OEM is a USB 2.0 high-speed, high-power device with a maximum theoretical transfer rate of 480 Mb/s. Using a high-speed FPGA-based architecture, the NI USB-8452 OEM supports SPI data acquisition up to 50 MHz and I^2C communication up to 3.3 MHz. The programmable reference voltage covers popular logic families from 1.2 V to 3.3 V, which makes the NI USB-8452 OEM versatile for most SPI/I^2C tests and verifications.

Refer to *Safety* in Appendix A, *NI USB-845x Hardware Specifications*, for important safety information.

Installing Software

Install the software provided with the NI USB-8452 OEM. Refer to the *NI-845x Software and Hardware Installation Guide* for more information.

Setting Up Hardware

Complete the following steps to set up the hardware:

1. Attach a suitable cable to the IDE-40 connector (pin 20 is left out on purpose) on the NI USB-8452 OEM module.

Note You can use a standard 40-pin IDE (ribbon) cable to access the front-end I/O pins (SPI, I^2C, and digital I/O) of the NI USB-8452 OEM module.

2. Connect the other end of the cable to your board. Refer to Figure 3-8 for the IDE connector pinout. Refer to *Signal Descriptions* for more information about the signals. Connect the ground pins next to the functional pins for better signal integrity.

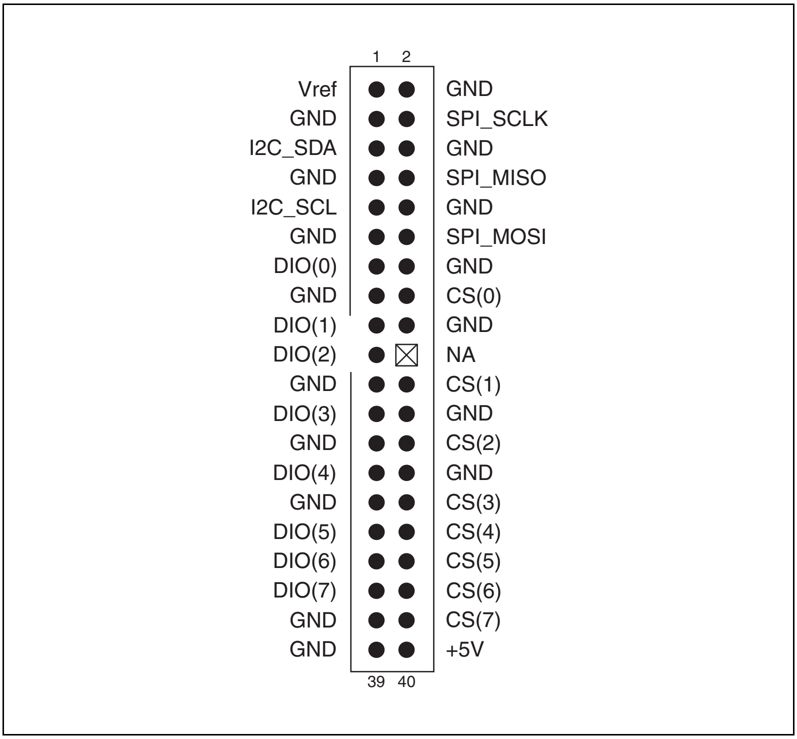


Figure 3-8. IDE-40 Connector Pin Assignments

3. The USB-8452 OEM board has a USB series B-type receptacle for connection to the host machine. Use a suitable cable to plug in the USB series B-type receptacle.

Signal Descriptions

Table 3-4 describes the signals available on the I/O connectors.

Table 3-4. Signal Descriptions

Signal Name	Direction	Description
SPI_SCLK	Output	**SPI Clock**—SPI output clock signal to slave devices capable of clock rates up to 50 MHz.
SPI_MOSI	Output	**Master Output Slave Input**—SPI communication signal to slave device.
SPI_MISO	Input	**Master Input Slave Output**—SPI communication signal from slave device.
CS<0..7>[1]	Output	**Chip Select Signals**—Outputs used to select the desired SPI peripheral device.
DIO<0..7>[2]	Input or Output	**Digital I/O Ports**—You can individually configure each signal as an input or push-pull output.
I2C_SCL	Open-drain[3]	**I^2C Clock**—I^2C clock signal to slave devices, capable of clock rates up to 3.3 MHz.
I2C_SDA	Open-drain[3]	**I^2C Serial Data**—Data signal for I^2C communication.
+5V	Output	**+5 V**—Fixed 5 V output with ±5% tolerance, with a maximum output drive capability of 20 mA.
Vref	Output	**Vref**—User programmable I^2C/SPI/DIO reference voltage output. Used for internal and external voltage reference. Maximum output drive capability of 20 mA.
GND	—	**Ground**—Ground reference for all IO interfaces and +5 V, Vref voltage references.
NA	—	Not available.

[1] You can configure CS(0) as hardware-timed chip-select, which has a fixed timing relationship to SPI signal lines. Refer to Chapter 11, *Using the NI-845x SPI Stream API*, for details.

[2] Some of these pins have special functionality in SPI stream mode. Refer to Chapter 11, *Using the NI-845x SPI Stream API*, for details.

[3] You can enable or disable onboard pull-up resistors. You must enable these for Vref ≤ 1.8 V for the FPGA to properly detect a low-to-high transition. Refer to Chapter 5, Using the NI-845x I2C API, for more information about enabling pull-ups on the I^2C lines.

Front-End I/O Interfaces

Caution Exceeding the maximum input voltage ratings or maximum output ratings, which are listed in Appendix A, *NI USB-845x Hardware Specifications*, can damage the USB device and the computer. National Instruments is not liable for any damage resulting from such signal connections.

SPI Interface

The NI USB-8452 OEM SPI master interface supports clock rates up to 50 MHz and can be divided down to support lower rates. Meanwhile, you also can switch voltage levels by configuring the programmable voltage regulator on board. The NI USB-8452 OEM supports logic families of 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V.

Figure 3-9 shows a typical SPI interface to three peripherals. All devices share the SPI MISO, SPI MOSI, and SPI CLK signals. Each peripheral has its own CS signal for addressing it.

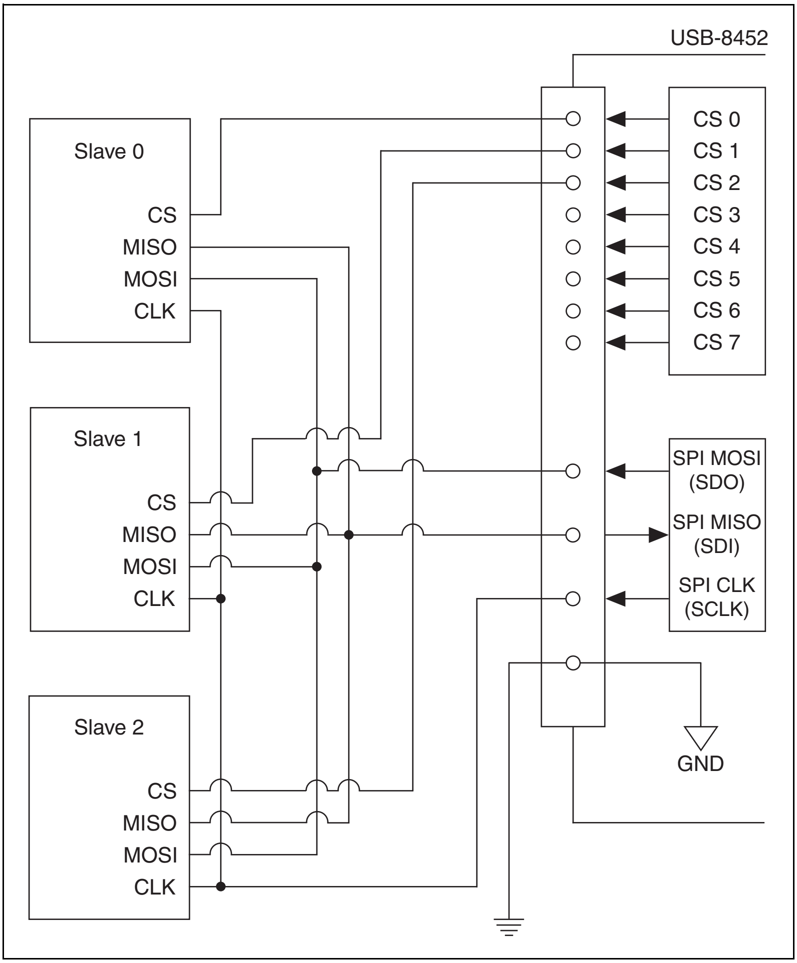


Figure 3-9. SPI Interface to Three Peripherals

The NI USB-8452 OEM SPI master interface supports two modes: standard mode and stream mode. The standard mode is generally backward compatible with the NI USB-8451 (except for programmable logic levels and clock rates). Meanwhile, in stream mode you have more control over SPI timing and packet formation. This mode supports hardware timed data streaming, which increases system throughput in cases of high-speed data acquisition.

Standard Mode

The SPI standard API provides the most fundamental SPI transaction type: write/read. You can access most existing SPI devices using this transaction. This mode is backward compatible with the NI USB-8451 and works with the NI-845*x* basic and advanced APIs. SPI packet length is fixed to 8 bits (1 byte).

Stream Mode

With the stream API, you can get direct control over SPI timing parameters and additional functional pins such as hardware timed chip select (CS(0)), data ready (DIO(1)), and conversion (DIO(0)) lines, which are widely adopted in modern analog to digital converters (ADCs). You can define output/trigger waveforms based on a 10 ns system clock and run continuously to stream in/out data. Refer to Chapter 11, *Using the NI-845x SPI Stream API*, for further information. You can combine standard and stream modes to generate a complete configuration and acquisition loop.

I^{2}C Interface

Figure 3-10 shows a typical I^{2}C interface to two peripherals. All devices on the I^{2}C bus share the SDA and SCL signals. SDA and SCL lines must be pulled up internally for 1.2 V, 1.5 V, and 1.8 V. SDA and SCL lines may be pulled up internally or externally for 2.5 V and 3.3 V. Refer to the I^{2}C specification to select the correct resistor values if using external pull-ups.

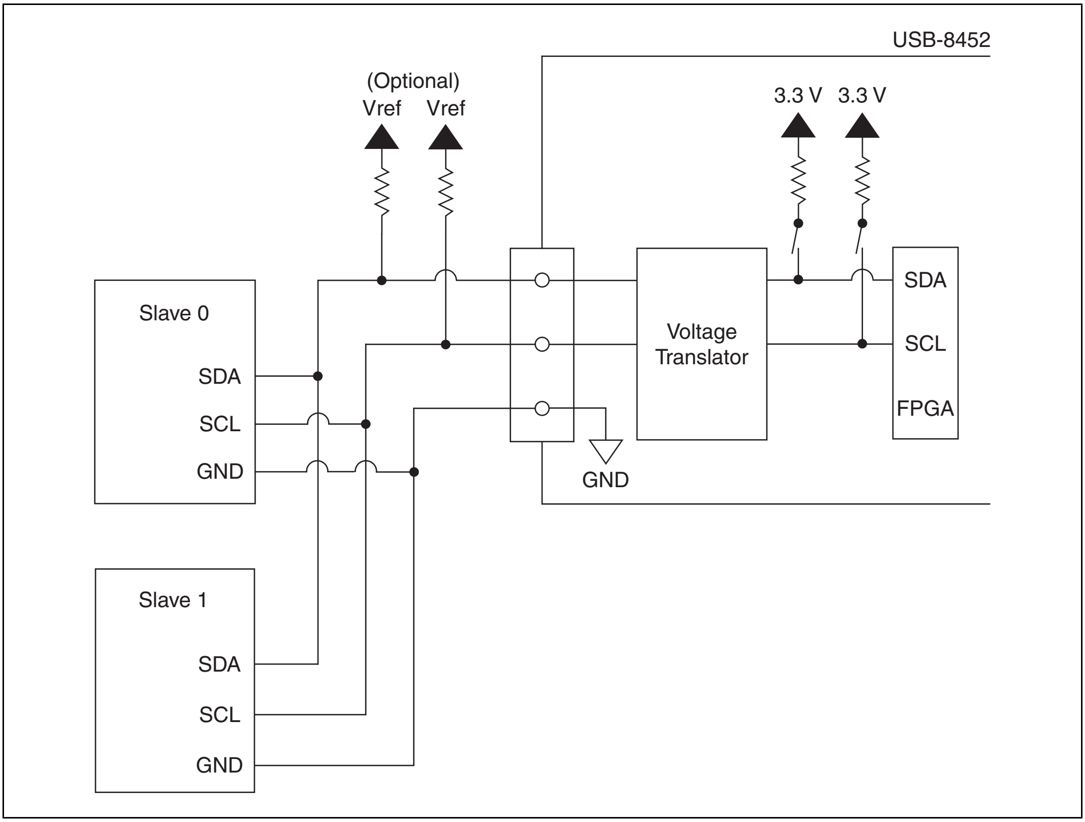


Figure 3-10. I²C Interface to Two Peripherals

The NI USB-8452 OEM I²C master interface supports Standard mode, Fast mode, Fast mode Plus, and High Speed mode (HS mode), defined in I²C 3.0 specifications. Refer to *I²C Interface* in Appendix A, *NI USB-845x Hardware Specifications*, for list of supported I²C data rates.

Refer to Chapter 5, *Using the NI-845x I2C API*, for more information about programming and using the I²C interface.

Digital I/O (DIO)

You can program each NI USB-8452 OEM DIO line individually as a static DI or DO line. You can use these I/O lines to monitor or control digital signals directly. You also can configure the logic level the same way as SPI and I²C interfaces. All samples of DI lines and updates of DO lines are software timed. All DIO lines are push-pull if configured as output. If disabled, these lines are tri-stated with weak pull-down resistors (40 kΩ).

Refer to Chapter 14, *Using the NI-845x DIO API*, for more information about programming and using the DIO lines.

Note While the device is in USB suspend, all I/O outputs are disabled.

LED Indicators

The NI USB-8452 OEM has two LED indicators alongside the USB connector, as shown in Figure 3-11.

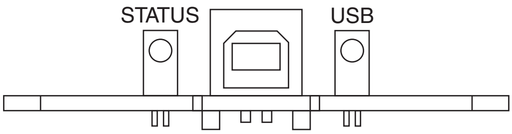


Figure 3-11. LED Indicators

The blue LED marked *USB* is the USB status LED.

State	Status
Off	Unplugged or suspend mode, or disabled
Solid blue	Connected to an active USB port

The green LED marked *STATUS* indicates the SPI/I^2C interface's current working status.

State	Status
Off	SPI/I^2C interface is idle
Blinking green	SPI/I^2C interface is active
Solid green	SPI interface is waiting on response from slave

I/O Protection

Each signal line is protected against overvoltage, undervoltage, and overcurrent conditions. However, you should avoid these fault conditions by following these guidelines:

- If you configure a line as an output, do not connect it to any external signal source, ground signal, or power supply.
- If you configure a line as an output, understand the current requirements of the load connected to these signals. Do not exceed the specified current output limits of the NI USB-8452 OEM.
- If you configure a line as an input, do not drive the line with voltages outside its normal operating range.
- Treat the NI USB-8452 OEM as you would treat any static sensitive device. Always properly ground yourself and the equipment when handling the USB device or connecting to it.

Power-On States

At system startup and resume from suspend, the hardware tri-states all IO ports including I^2C, SPI, DIO, and CS lines, among which SPI, DIO and CS lines are weakly pulled down to GND with 40 kΩ resistors. The NI USB-8452 OEM does not drive any of the signals high or low.

Power Sources

+5 V Power Source

The NI USB-8452 OEM offers a 5 V output from pin 40. The voltage source is generated from an onboard regulator, with ±5% tolerance. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for more information. You can use this source to power external components with low power budget at 20 mA current maximum.

Note The +5 V power source output is enabled on the first NI-845*x* API call. While the device is in USB suspend, the +5 V power source output is disabled. The +5 V power source output is reenabled after the next NI-845*x* API call.

Caution If you accidentally short the +5 V source or apply an external load that exceeds the power budget, the NI USB-8452 OEM automatically enters over current protection and cuts off front power. In this case, you are warned to check your connection and reboot the system. In the meantime, front I/O activity is stopped.

Vref I/O Reference Voltage

The NI USB-8452 OEM also provides a programmable reference voltage from pin 1. You can configure this reference voltage as 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V. You can program the reference voltage based on the application, and the NI USB-8452 OEM board adapts to the programmed voltage level. The voltage source is provided mainly as a voltage reference to external circuitry or as power source for low power budget components. It can source 20 mA current maximum. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for more information.

Note Vref is not enabled before you choose a specific voltage or use the default voltage (3.3 V). While the device is in USB suspend, this output is disabled.

Caution If you accidentally short Vref or apply an external load that exceeds the power budget, the NI USB-8452 OEM automatically enters over current protection and cuts off front power. In this case, you are warned to check your connection and reboot the system. In the meantime, front I/O activity is stopped and tri-stated with weak pull down (40 kΩ) to GND.

4

Using the NI-845*x* API

The NI-845*x* API consists of handles (references), property nodes (LabVIEW only), and functions. A handle identifies a particular piece of hardware or the configuration for use in the API functions. For example, to access an NI 845*x* device, you first must create a device handle by providing the name of the NI 845*x* device configured in Measurement & Automation Explorer (MAX). After creating the device handle, the NI-845*x* software functions use the returned handle to determine which NI 845*x* device to communicate with.

The NI-845*x* API has other handles also. An example is a configuration handle that describes the device characteristics used for communication. An I²C configuration contains properties such as the bus clock rate and device address to use for communication. Refer to the specific API calls for more information on how to use handles in the NI-845*x* API. In LabVIEW, you can pass the configuration handle into a property node to configure specific characteristics. In other languages, you pass the handle into the special configuration functions to configure the characteristics. In addition, many API functions use the configuration to perform the desired action.

5

Using the NI-845*x* I^{2}C API

This chapter helps you get started with the I^{2}C API.

I^{2}C Basic Programming Model

The I^{2}C Basic API provides the most fundamental I^{2}C transaction types: write, read, and write/read. You can access the majority of off-the-shelf I^{2}C devices using these transactions. The I^{2}C Basic API allows you to easily and quickly develop applications to communicate with these devices. For those situations in which the I^{2}C Basic API does not provide the functionality you need, use the I^{2}C Advanced API to create custom I^{2}C transactions.

When you use the I^{2}C Basic API, the first step is to create an I^{2}C configuration to describe the communication requirements between the NI 845*x* device and the I^{2}C slave device. To make an I^{2}C configuration, create an I^{2}C configuration reference and set the appropriate properties as desired. You can then read or write data to the I^{2}C slave device.

The diagram in Figure 5-1 describes the programming model for the NI-845*x* I^{2}C Basic API. Within the application, you repeat this programming model for each I^{2}C device. The diagram is followed by a description of each step in the model.

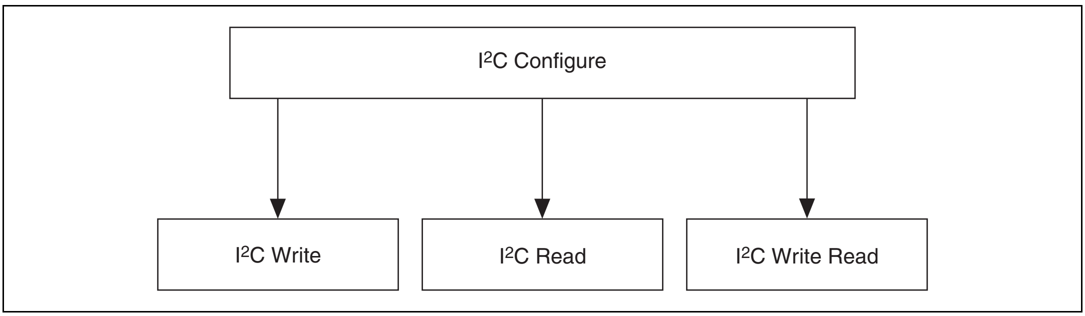


Figure 5-1. Basic Programming Model for I^{2}C Communication

I²C Configure

Use the **NI-845x I2C Configuration Property Node** in LabVIEW and `ni845xI2cConfiguration*` calls in other languages to set the specific I²C configuration that describes the characteristics of the device to communicate with.

I²C Write

Use **NI-845x I2C Write.vi** in LabVIEW and `ni845xI2cWrite` in other languages to write an array of data to an I²C slave device.

I²C Read

Use **NI-845x I2C Read.vi** in LabVIEW and `ni845xI2cRead` in other languages to read an array of data from an I²C slave device.

I²C Write Read

Use **NI-845x I2C Write Read.vi** in LabVIEW and `ni845xI2cWriteRead` in other languages to write an array of data followed by a read (combined format) on an I²C slave device.

I²C Advanced Programming Model

The NXP I²C specification is extremely flexible and allows multiple possibilities for constructing transactions beyond those handled by the I²C Basic API. The I²C Advanced API provides a set of script commands that allow you great flexibility in creating custom I²C transactions for your particular needs. For example, you can use scripting in the following scenarios:

- Validating a new device design, when you want to issue individual I²C conditions to the bus, with or without variable delays in between, so that you can observe device response.
- Issuing a transaction to a device and measuring its responses (using NI 845*x* DIO pins configured for input) at multiple points within the transaction.
- Using the NI 845*x* DIO pins configured for output to provide additional control or addressing.
- Doing performance testing, in which you see how a device responds to variable delays, clock rate changes, etc. within a transaction.
- Issuing multiple reads and writes to a device, or multiple devices, within one transaction, to avoid relinquishing the bus.

When you use the I²C Advanced API, the first step is to create a script that describes the communication between an I²C master and an I²C slave device. Then you execute the script and extract the read data if needed. The script size is limited only by the amount of memory available on your PC. The number of read commands, I2C Script Read, I2C Script DIO Read Port, and I2C Script DIO Read Line within each script is limited to 64.

The diagram in Figure 5-2 describes an example of programming with the scripting functions for the NI-845*x* I²C Advanced API. The diagram is followed by a description of each step in the model.

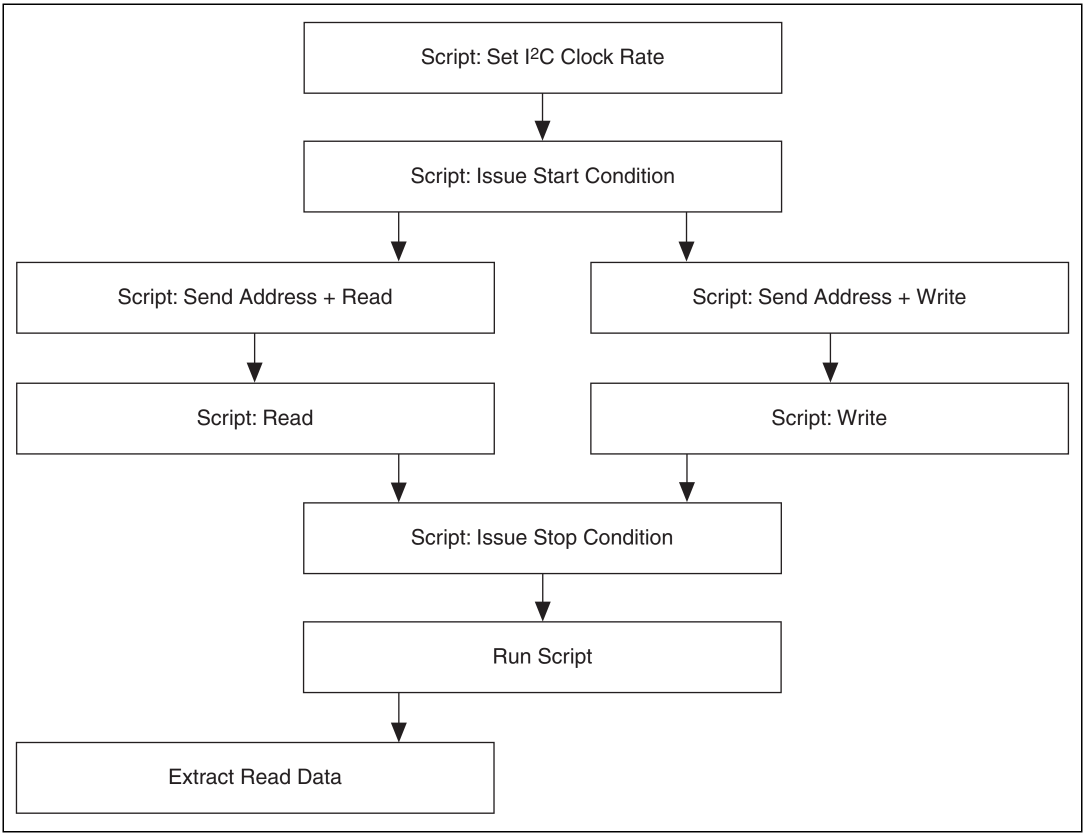


Figure 5-2. Example of Advanced Programming Model with Scripting API for I²C Communication

Script: Set I^{2}C Clock Rate

Use **NI-845x I2C Script Clock Rate.vi** in LabVIEW and `ni845xI2cScriptClockRate` in other languages to add an I^{2}C Script Clock Rate command to the I^{2}C script. This command sets the I^{2}C clock rate for the I^{2}C port you specify when you run the script.

Script: Pullup Enable

Use **NI-845x I2C Script Pullup Enable.vi** in LabVIEW and `ni845xI2cScriptPullupEnable` in other languages to add an I^{2}C Script Pullup Enable command to the I^{2}C script. This command enables or disables the internal I^{2}C pullup resistors. This command is valid only for NI 845*x* devices with onboard pull-up resistors.

Script: Set I2C High Speed Clock Rate

Use **NI-845x I2C Script High Speed Clock Rate.vi** in LabVIEW and `ni845xI2cScriptHsClockRate` in other languages to add an I^{2}C Script HS Clock Rate command to the I^{2}C script. This command sets the I^{2}C High Speed clock rate for the I^{2}C port you specify when you run the script. This command is valid only for NI 845*x* devices that support High Speed I^{2}C.

Script: Set I2C High Speed Enable

Use **NI-845x I2C Script HS Enable.vi** in LabVIEW and `ni845xI2cScriptHsEnable` in other languages to add an I^{2}C Script HS Enable command to the I^{2}C script. This command enables or disables High Speed mode. This command is valid only for NI 845*x* devices that support High Speed I^{2}C.

Script: Issue Start Condition

Use **NI-845x I2C Script Issue Start.vi** in LabVIEW and `ni845xI2cScriptIssueStart` in other languages to add an I^{2}C Script Issue Start command to the I^{2}C script. This command issues a start condition on the I^{2}C bus connected to the I^{2}C port you specify when you run the script.

Script: Send High Speed Master Code

Use **NI-845x I2C Script Master Code.vi** in LabVIEW and `ni845xI2cScriptHsMasterCode` in other languages to add an I²C Script HS Master Code command to the I²C script. This command transmits the I²C High Speed master code. This command is valid only for NI 845*x* devices that support High Speed I²C.

Script: Send Address + Read

Use **NI-845x I2C Script Address+Read.vi** in LabVIEW and `ni845xI2cScriptAddressRead` in other languages to add an I²C Script Address+Read command to the I²C script. This command writes a 7-bit address, followed by the direction bit set to read, to the I²C bus connected to the I²C port you specify when you run the script.

Script: Read

Use **NI-845x I2C Script Read.vi** in LabVIEW and `ni845xI2cScriptRead` in other languages to add an I²C Script Read command to the I²C script. This command reads an array of data from a device connected to the I²C port you specify when you run the script.

Script: Send Address + Write

Use **NI-845x I2C Script Address+Write.vi** in LabVIEW and `ni845xI2cScriptAddressRead` in other languages to add an I²C Script Address+Write command to the I²C script. This command writes a 7-bit address, followed by the direction bit set to write, to the I²C bus connected to the I²C port you specify when you run the script.

Script: Write

Use **NI-845x I2C Script Write.vi** in LabVIEW and `ni845xI2cScriptWrite` in other languages to add an I²C Script Write command to the I²C Script. This command writes an array of data to an I²C slave device when you run the script.

Script: Issue Stop Condition

Use **NI-845x I2C Script Issue Stop.vi** in LabVIEW and `ni845xI2cScriptIssueStop` in other languages to add an I²C Script Issue Stop command to the I²C script. This command issues a stop condition on the I²C bus connected to the I²C port you specify when you run the script.

Run Script

Use **NI-845x I2C Run Script.vi** in LabVIEW and `ni845xI2cScriptRun` in other languages to execute an I²C script on the desired device.

Extract Read Data

Use **NI-845x I2C Extract Script Read Data.vi** in LabVIEW and `ni845xI2cScriptExtractReadData` in other languages to extract the desired read data from an I²C script that has been previously run. Each I²C script read command (I2C Script Read, I2C Script DIO Read Port, I2C Script DIO Read Line) returns a script read index to be passed into the Extract Read Data function.

6

NI-845*x* I^2C API for LabVIEW

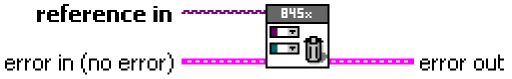
This chapter lists the LabVIEW VIs for the NI-845*x* I^2C API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

General Device

NI-845x Close Reference.vi

Purpose

Closes a previously opened reference.



Inputs

reference in is a reference to an NI 845*x* device, I²C configuration, SPI configuration, SPI stream configuration, I²C script, or SPI script.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

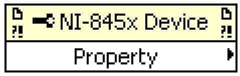
Description

Use **NI-845x Close Reference.vi** to close a previously opened reference.

NI-845x Device Property Node

Purpose

A property node with the NI-845*x* Device class preselected. This property node allows you to modify properties of your NI 845*x* device.



Inputs

device reference in is a reference to an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to an NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

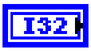
source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x Device Property Node**.

DIO:Active Port

The **DIO:Active Port** property sets the active DIO port for further DIO port configuration. The format for this property is a decimal string. For example, the string `0` represents DIO Port 0. The default value of this property is 0. For NI 845*x* devices with one DIO port, the port value must be 0.



DIO:Driver Type

The **DIO:Driver Type** property configures the active DIO port with the desired driver type characteristics. **DIO:Driver Type** uses the following values:

`Open-Drain`

The DIO driver type is configured for open-drain.

`Push-Pull`

The DIO driver type is configured for push-pull. The actual voltage driven (when sourcing a high value) is determined by the *I/O Voltage Level* property.

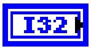
The default value of this property is `Push-Pull`.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available driver types on your hardware.

DIO:Line Direction Map

The **DIO:Line Direction Map** property sets the line direction map for the active DIO Port. The value is a bitmap that specifies the function of each individual line within the port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

The default value of this property is `0` (all lines configured for input).



I/O Voltage Level

The **I/O Voltage Level** property sets the board voltage. This property sets the voltage for SPI, I^{2}C, and DIO. The default value for this property is `3.3V`. This property uses the following values:

`3.3V`

I/O Voltage is set to 3.3 V.

`2.5V`

I/O Voltage is set to 2.5 V.

`1.8V`

I/O Voltage is set to 1.8 V.

`1.5V`

I/O Voltage is set to 1.5 V.

`1.2V`

I/O Voltage is set to 1.2 V.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

I^{2}C Pullup Enable

The **I^{2}C Pullup Enable** property enables or disables the internal pullup resistors connected to SDA and SCL.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your hardware has onboard pull-up resistors.

NI-845x Device Reference

Purpose

Specifies the device resource to be used for communication.

Description

Use the **NI-845x Device Reference** to describe the NI 845*x* device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845*x* API call to invoke the function on the associated NI 845*x* device.

Configuration

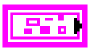
NI-845x I2C Configuration Property Node

Purpose

A property node with the NI-845*x* I^{2}C Configuration class preselected. This property node allows you to query and modify I^{2}C configuration properties of your NI 845*x* device.

Inputs

i2c configuration in is a reference to a specific I^{2}C configuration that describes the characteristics of the device to communicate with.



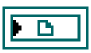
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the error in cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

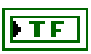
source identifies the VI where the error occurred.

Outputs

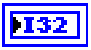


i2c configuration out is a reference to a specific I^{2}C configuration that describes the characteristics of the device to communicate with.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x I2C Configuration Property Node**.

Port

Specifies the I^2C port that this configuration communicates across.

Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the number of I^2C ports your NI 845*x* device supports.

The default value of this property is `0`.

Clock Rate in kHz

Specifies the I^2C clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports. If your hardware does not support the supplied clock rate, a warning is generated, and the next smallest supported clock rate is used. If the supplied clock rate is smaller than the smallest supported clock rate, an error is generated.

If High Speed mode is enabled, this clock rate is used to transfer the master code.

The default value of this property is `100` kHz.

Address Size

Specifies the addressing scheme to use when addressing the I^2C slave device this configuration describes. **Address Size** uses the following values:

`7 Bits`

The NI 845*x* hardware uses the standard 7-bit addressing when communicating with the I^2C slave device.

`10 Bits`

The NI 845*x* hardware uses the extended 10-bit addressing when communicating with the I²C slave device.

The default value of this property is `7 Bits`.

Address

Specifies the I²C slave address. The default address is `0`. For 7-bit device addressing, the NXP I²C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I²C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845*x* software follows the convention used in the NXP I²C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845*x* software internally sets the direction bit to the correct value, depending on the function (write or read). Some manufacturers specify the address for their 7-bit device as a byte. In such cases, bits 7–1 contain the slave address, and bit 0 contains the direction. When using the NI-845*x* software, discard the direction bit and right-shift the byte value by one to create the 7-bit address.

HighSpeed:Enable

Enables High Speed (HS) mode. The default is set to High Speed mode disabled. When High Speed mode is enabled, the NXP I²C Specification defines a Master Code and a High Speed clock rate.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your NI 845*x* device supports High Speed mode.

HighSpeed:ClockRate

Specifies the I²C clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which High Speed clock rates your NI 845*x* device supports. If your hardware does not support the supplied High Speed clock rate, a warning is generated, and the next smallest supported High Speed clock rate is used. If the supplied High Speed clock rate is smaller than the smallest supported High Speed clock rate, an error is generated.

The default value of this property is 1700 kHz.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the High Speed clock rates your NI 845*x* device supports.

HighSpeed:MasterCode

Specifies the master code to be used for High Speed mode. The NXP I^2C Specification defines the master code as a 3-bit number that is unique on the I^2C bus.

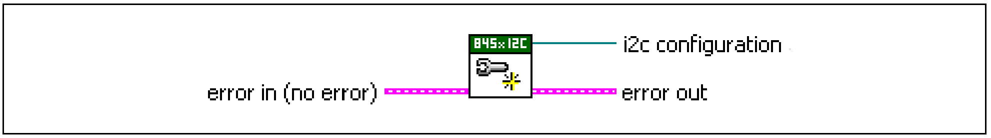
This property requires High Speed mode to be enabled.

The default value of this property is `1`.

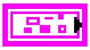
NI-845x I2C Create Configuration Reference.vi

Purpose

Creates a new NI-845x I^{2}C configuration.

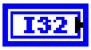


Inputs



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

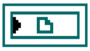
status is TRUE if an error occurred. This VI is not executed when status is TRUE.



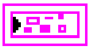
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

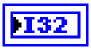


i2c configuration is a reference to the newly created NI-845x I^{2}C configuration.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

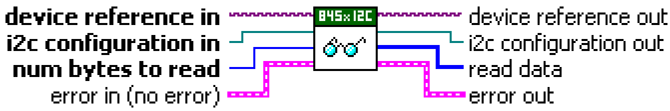
Use **NI-845x I2C Create Configuration Reference.vi** to create a new configuration to use with the NI-845*x* I²C Basic API. Pass the reference to a property node to make the configuration match the settings of your I²C slave. Then, pass the configuration to the I²C basic functions to execute them on the described I²C slave. After you finish communicating with your I²C slave, pass the reference into a new property node to reconfigure it or use **NI-845x Close Reference.vi** to delete the configuration.

Basic

NI-845x I2C Read.vi

Purpose

Reads an array of data from an I²C slave device.

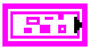


Inputs

device reference in is a reference to an NI 845*x* device.

i2c configuration in is a reference to a specific I²C configuration that describes the characteristics of the device to communicate with. Connect this configuration reference into a property node to set the specific configuration parameters.

num bytes to read specifies the number of bytes to read from the I²C slave.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

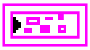
source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

i2c configuration out is a reference to the I^{2}C configuration after this VI runs.

read data contains an array of read data from the I^{2}C slave.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

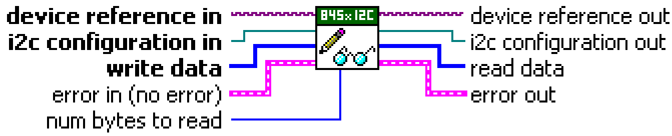
Use **NI-845x I2C Read.vi** to read an array of data from an I^{2}C slave device. Per the NXP I^{2}C Specification, each byte read up to the last byte is acknowledged. The last byte is not acknowledged. This VI first waits for the I^{2}C bus to be free. If the I^{2}C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7 or 10-bit I^{2}C read transaction, per the NXP I^{2}C Specification. The address type (7 or 10-bit) and other configuration parameters are specified by the configuration wired into **i2c configuration in**. If the NI 845*x* device tries to access the bus at the same time as another I^{2}C master device and loses arbitration, the read transaction is terminated and an error is returned. If the address of the transaction is not acknowledged by the slave device, an error is returned. Otherwise, the transaction is completed, and a stop condition is generated per the NXP I^{2}C Specification.

Before using **NI-845x I2C Read.vi**, you need to ensure that the configuration parameters specified in **i2c configuration in** are correct for the device you want to access.

NI-845x I2C Write Read.vi

Purpose

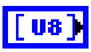
Performs a write followed by read (combined format) on an I²C slave device.



Inputs

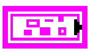
device reference in is a reference to an NI 845*x* device.

i2c configuration in is a reference to a specific I²C configuration that describes the characteristics of the device to communicate with. Connect this configuration reference into a property node to set the specific configuration parameters.



write data contains an array of data to write to the I²C slave.

num bytes to read specifies the number of bytes to read from the I²C slave.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

i2c configuration out is a reference to the I^{2}C configuration after this VI runs.

read data contains an array of read data from the I^{2}C slave.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x I2C Write Read.vi** to perform a write followed by read (combined format) on an I^{2}C slave device. During the read portion of the transaction, per the NXP I^{2}C Specification, each byte read up to the last byte is acknowledged. The last byte is not acknowledged. This VI first waits for the I^{2}C bus to be free. If the I^{2}C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7 or 10-bit I^{2}C write/read transaction. Per the NXP I^{2}C Specification, the write/read transaction consists of a start–write–restart–read– stop sequence.

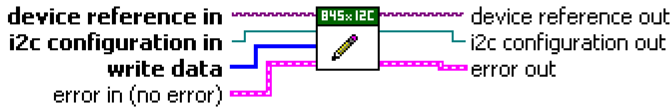
The address type (7 or 10-bit) and other configuration parameters are specified by the configuration wired into **i2c configuration in**. If the NI 845*x* device tries to access the bus at the same time as another I^{2}C master device and loses arbitration, the read transaction is terminated and an error is returned. If an address or byte write within the transaction is not acknowledged by the slave device, an error is returned. Otherwise, the transaction is completed and a stop condition is generated per the NXP I^{2}C Specification. It should be noted that this type of combined transaction is provided because it is commonly used (for example, with EEPROMs). The NXP I^{2}C Specification provides flexibility in the construction of I^{2}C transactions. The NI-845*x* I^{2}C scripting VIs allow creating and customizing complex I^{2}C transactions as needed.

Before using **NI-845x I2C Write Read.vi**, you need to ensure that the configuration parameters specified in **i2c configuration in** are correct for the device you want to access.

NI-845x I2C Write.vi

Purpose

Writes an array of data to an I^2C slave device.

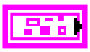


Inputs

device reference in is a reference to an NI 845*x* device.

i2c configuration in is a reference to a specific I^2C configuration that describes the characteristics of the device to communicate with. Connect this configuration reference into a property node to set the specific configuration parameters.

write data contains an array of data to write to the I^2C slave.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

i2c configuration out is a reference to the I^{2}C configuration after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x I2C Write.vi** to write an array of data to an I^{2}C slave device. This VI first waits for the I^{2}C bus to be free. If the I^{2}C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7 or 10-bit I^{2}C write transaction, per the NXP I^{2}C Specification. The address type (7 or 10-bit) and other configuration parameters are specified by the configuration wired into **i2c configuration in**. If the NI 845*x* device tries to access the bus at the same time as another I^{2}C master device and loses arbitration, the write transaction is terminated and an error is returned. If any byte of the transaction is not acknowledged by the slave device, an error is returned. Otherwise, the transaction is completed, and a stop condition is generated per the NXP I^{2}C Specification.

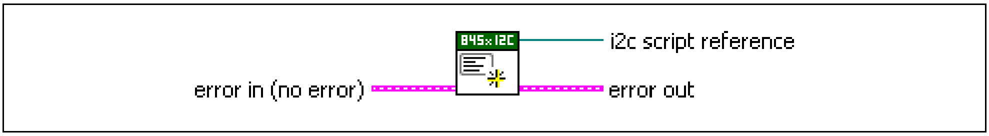
Before using **NI-845x I2C Write.vi**, you need to ensure that the configuration parameters specified in **i2c configuration in** are correct for the device you currently want to access.

Advanced

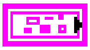
NI-845x I2C Create Script Reference.vi

Purpose

Creates a new NI-845*x* I²C script.



Inputs



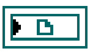
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

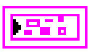
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

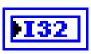


i2c script reference is a reference to the newly created NI-845*x* I²C script.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

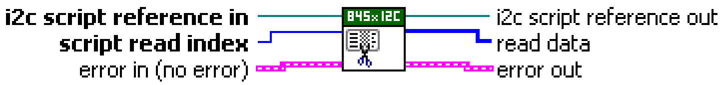
Description

Use **NI-845x I2C Create Script Reference.vi** to create a new script to use with the NI-845*x* I^{2}C Advanced API. Pass the reference to I^{2}C script functions to create the script. Then, call **NI-845x I2C Run Script.vi** to execute your script on your NI 845*x* device. After you finish executing your script, use **NI-845x Close Reference.vi** to delete the script.

NI-845x I2C Extract Script Read Data.vi

Purpose

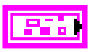
Extracts the desired read data from an I^{2}C script, referenced by **i2c script reference in**, which has been processed by **NI-845x I2C Run Script.vi**. Each script read command (**NI-845x I2C Script Read.vi**, **NI-845x I2C Script DIO Read Port.vi**, **NI-845x I2C Script DIO Read Line.vi**) returns a script read index. Data may be extracted for each script read index in a script, by wiring each to a separate **NI-845x I2C Extract Script Read Data.vi**.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

script read index identifies the read in the script whose data should be extracted.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

read data is the data returned for the script command specified by **script read index**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

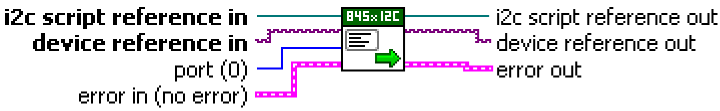
Use **NI-845x I2C Extract Script Read Data.vi** to extract the desired read data from an I^{2}C script, referenced by **i2c script reference in**, which has been processed by **NI-845x I2C Run Script.vi**. Each I^{2}C script read command (**NI-845x I2C Script Read.vi**, **NI-845x I2C Script DIO Read Port.vi, NI-845x I2C Script DIO Read Line.vi**) returns a script read index.

Data may be extracted for each script read in different ways. For example, you can wire the script read index output of each script read VI to its own **NI-845x I2C Extract Script Read Data.vi**. You can also place **NI-845x I2C Extract Script Read Data.vi** in a For Loop and wire the loop iteration terminal to the **script read index** input. Add one to the script read index output of the last read and wire this value to the loop count terminal. The output of the For Loop will be an array of read data arrays.

NI-845x I2C Run Script.vi

Purpose

Executes an I²C script referenced by **i2c script reference in** on the device referenced by **device reference in**.

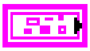


Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

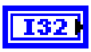
device reference in is a reference to an NI 845*x* device.

port specifies the I²C port this script runs on.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

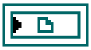
status is TRUE if an error occurred. This VI is not executed when status is TRUE.



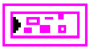
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

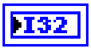
Outputs


i2c script reference out is a reference to the I^{2}C script after this VI runs.


device reference out is a reference to the NI 845*x* device after this VI runs.


error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.


status is TRUE if an error occurred.


code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.


source identifies the VI where the error occurred.

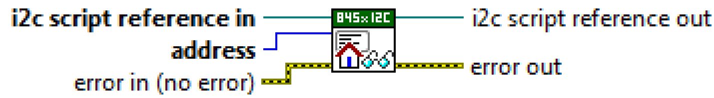
Description

Use **NI-845x I2C Run Script.vi** to execute an I^{2}C script referenced by **i2c script reference in** on the device referenced by **device reference in**. You must first create an I^{2}C script using the I^{2}C scripting VIs. Next, you wire its script reference into **i2c script reference in**. If you have multiple NI 845*x* devices installed in your system, you can select which device to write your I^{2}C script to by wiring its device reference to **device reference in**. If your NI 845*x* device supports multiple I^{2}C ports, you can also select which port to write your I^{2}C script to. For single I^{2}C port NI 845*x* devices, you must use the default port (`0`). In this way, you can create one script to run on various NI 845*x* devices, on various I^{2}C ports within those devices. **NI-845x I2C Run Script.vi** loads and executes your I^{2}C script on the NI 845*x* device and I^{2}C port you specify, then returns success or error. If your script contained any read commands, you may use **NI-845x I2C Extract Script Read Data.vi** to extract the read data after executing **NI-845x I2C Run Script.vi**.

NI-845x I2C Script Address+Read.vi

Purpose

Adds an I²C Script Address+Read command to an I²C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I²C bus. The direction bit is internally set to `1` for read.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

address specifies the 7-bit address to read. For 7-bit device addressing, the NXP I²C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I²C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845*x* software follows the convention used in the NXP I²C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845*x* software internally sets the direction bit to the correct value, depending on the function (write or read). Some manufacturers specify the address for their 7-bit device as a byte. In such cases, bits 7–1 contain the slave address, and bit 0 contains the direction. When using the NI-845*x* software, discard the direction bit and right-shift the byte value by one to create the 7-bit address.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^2C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

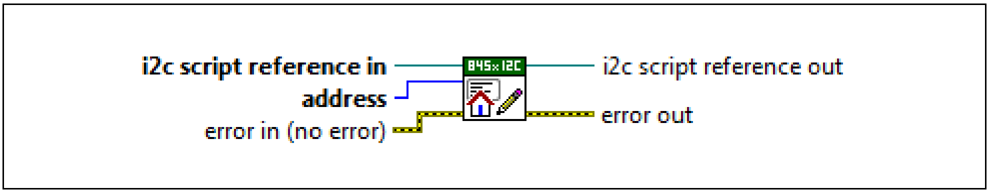
Description

Use **NI-845x I2C Script Address+Read.vi** to add an I^2C Script Address+Read command to an I^2C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I^2C bus connected to the I^2C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The direction bit is internally set to `1` for read. This command assumes that a start condition has been previously issued to the I^2C bus using an I^2C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I^2C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, **NI-845x I2C Run Script.vi** exits with an error.

NI-845x I2C Script Address+Write.vi

Purpose

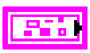
Adds an I^{2}C Script Address+Write command to an I^{2}C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I^{2}C bus. The direction bit is internally set to `0` for write.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

address specifies the 7-bit address to write. For 7-bit device addressing, the NXP I^{2}C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I^{2}C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845*x* software follows the convention used in the NXP I^{2}C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845*x* software internally sets the direction bit to the correct value, depending on the function (write or read). Some manufacturers specify the address for their 7-bit device as a byte. In such cases, bits 7–1 contain the slave address, and bit 0 contains the direction. When using the NI-845*x* software, discard the direction bit and right-shift the byte value by one to create the 7-bit address.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

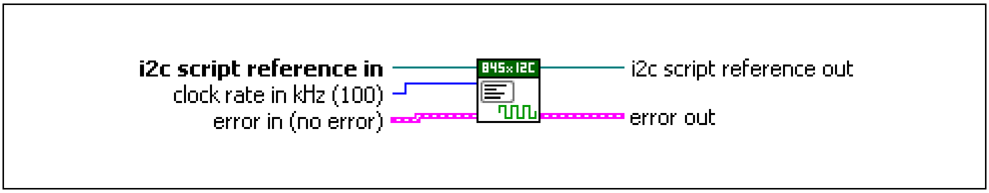
Description

Use **NI-845x I2C Script Address+Write.vi** to add an I^{2}C Script Address+Write command to an I^{2}C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I^{2}C bus connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The direction bit is internally set to `0` for write. This command assumes that a start condition has been previously issued to the I^{2}C bus using an I^{2}C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I^{2}C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, **NI-845x I2C Run Script.vi** exits with an error.

NI-845x I2C Script Clock Rate.vi

Purpose

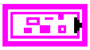
Adds an I^{2}C Script Clock Rate command to an I^{2}C script referenced by **i2c script reference in**. This command sets the I^{2}C clock rate.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

clock rate in kHz specifies the I^{2}C clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports.



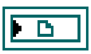
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

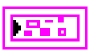
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



i2c script reference out is a reference to the I^{2}C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

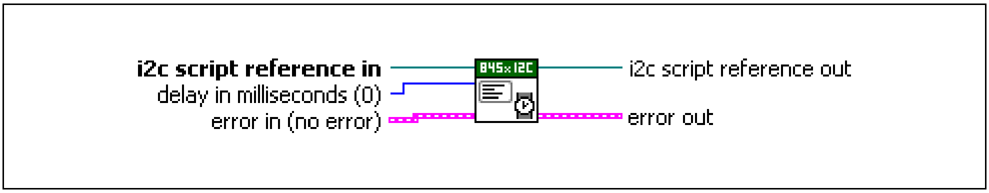
Description

Use **NI-845x I2C Script Clock Rate.vi** to add an I^{2}C Script Clock Rate command to an I^{2}C script referenced by **i2c script reference in**. This command sets the I^{2}C clock rate for the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845*x* driver adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

NI-845x I2C Script Delay.vi

Purpose

Adds an I^{2}C Script Delay command to an I^{2}C script referenced by **i2c script reference in**. This command adds a delay after the previous I^{2}C script command.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

delay in milliseconds specifies the desired delay.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

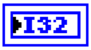
source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

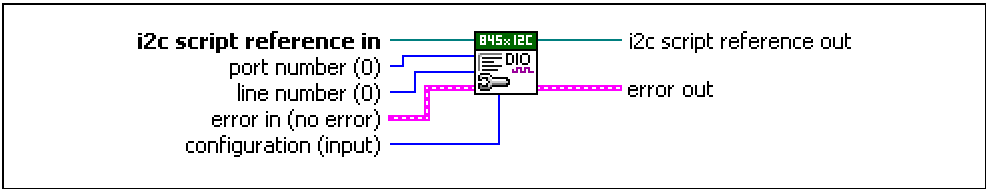
Description

Use **NI-845x I2C Script Delay.vi** to add an I^2C Script Delay command to an I^2C script referenced by **i2c script reference in**. This command adds a delay after the previous I^2C script command.

NI-845x I2C Script DIO Configure Line.vi

Purpose

Adds an I²C Script DIO Configure Line command to an I²C script referenced by **i2c script reference in**. This command configures a DIO line on an NI 845*x* device.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to configure.

configuration specifies the line configuration. **configuration** uses the following values:

input	The line is configured for input.
output	The line is configured for output.

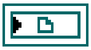
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

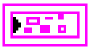
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



i2c script reference out is a reference to the I²C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

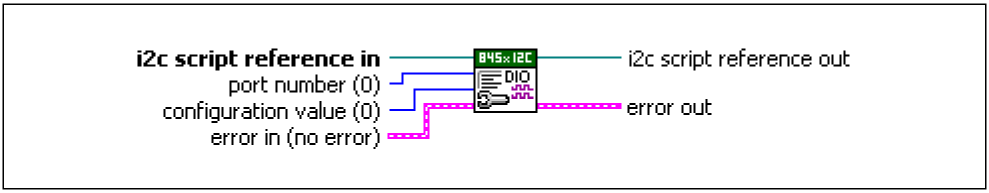
Description

Use **NI-845x I2C Script DIO Configure Line.vi** to add an I²C Script DIO Configure Line command to an I²C script referenced by **i2c script reference in**. This command allows you to configure one line, specified by **line number**, of a byte-wide DIO port, as an input or output. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x I2C Script DIO Configure Port.vi

Purpose

Adds an I²C Script DIO Configure Port command to an I²C script referenced by **i2c script reference in**. This command configures a DIO port on an NI 845*x* device.

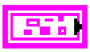


Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

port number specifies the DIO port to configure.

configuration value is a bitmap that specifies the function of each individual line of a port. If bit *x* = 1, line *x* is an output. If bit *x* = 0, line *x* is an input.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

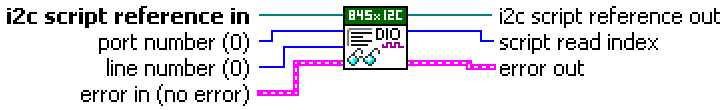
Description

Use **NI-845x I2C Script DIO Configure Port.vi** to add an I^{2}C Script DIO Configure Port command to an I^{2}C script referenced by **i2c script reference in**. This command allows you to configure all eight lines of a byte-wide DIO port. Setting a bit to `1` configures the corresponding DIO port line for output. Setting a bit to `0` configures the corresponding port line for input. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the port to configure. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x I2C Script DIO Read Line.vi

Purpose

Adds an I²C Script DIO Read Line command to an I²C script referenced by **i2c script reference in**. This command reads from a DIO line on an NI 845*x* device.

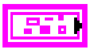


Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

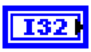
port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

script read index is the index of the read command within the script. It is used as an input into **NI-845x I2C Extract Script Read Data.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

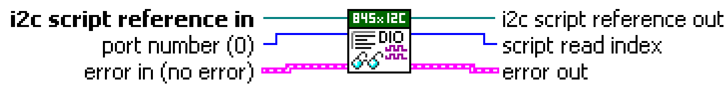
Use **NI-845x I2C Script DIO Read Line.vi** to add an I^{2}C Script DIO Read command to an I^{2}C script referenced by **i2c script reference in**. This command allows you to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

To obtain the logic level read from the specified DIO port line, wire **script read index** to **NI-845x I2C Extract Script Read Data.vi** after script execution. If **NI-845x I2C Extract Script Read Data.vi** returns `0`, the logic level read on the specified line was low. If **NI-845x I2C Extract Script Read Data.vi** returns `1`, the logic level read on the specified line was high.

NI-845x I2C Script DIO Read Port.vi

Purpose

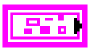
Adds an I²C Script DIO Read Port command to an I²C script referenced by **i2c script reference in**. This command reads from a DIO port on an NI 845*x* device.



Inputs

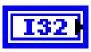
i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

port number specifies the DIO port to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

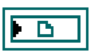
status is TRUE if an error occurred. This VI is not executed when status is TRUE.



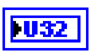
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

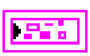
Outputs



i2c script reference out is a reference to the I²C script after this VI runs.



script read index is the index of the read command within the script. It is used as an input into **NI-845x I2C Extract Script Read Data.vi**.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

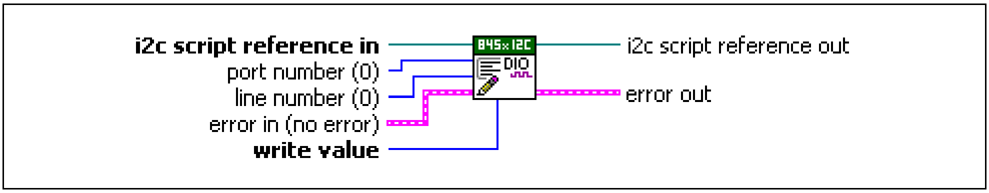
Use **NI-845x I2C Script DIO Read Port.vi** to add an I²C Script DIO Read Port command to an I²C script referenced by **i2c script reference in**. This command allows you to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

To obtain the data byte read from the specified DIO port, wire **script read index** to **NI-845x I2C Extract Script Read Data.vi** after script execution, which returns the data byte read by this script command.

NI-845x I2C Script DIO Write Line.vi

Purpose

Adds an I²C Script DIO Write Line command to an I²C script referenced by **i2c script reference in**. This command writes to a DIO line on an NI 845*x* device.

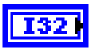


Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

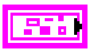
port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to write.



write value specifies the value to write to the line. **write value** uses the following values:

`0` (Logic Low)	The line is set to the logic low state.
`1` (Logic High)	The line is set to the logic high state.



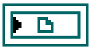
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

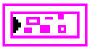
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



i2c script reference out is a reference to the I^{2}C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

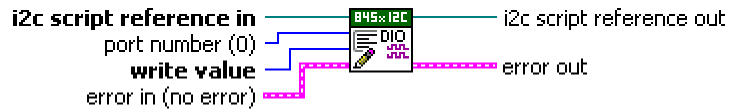
Description

Use **NI-845x I2C Script DIO Write Line.vi** to add an I^{2}C Script DIO Write Line command to an I^{2}C script referenced by **i2c script reference in**. This command allows you to write one line, specified by **line number**, of a byte-wide DIO port. If **write value** is `1`, the specified line's output is driven to a high logic level. If **write value** is `0`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x I2C Script DIO Write Port.vi

Purpose

Adds an I^2C Script DIO Write Port command to an I^2C script referenced by **i2c script reference in**. This command writes to a DIO port on an NI 845*x* device.

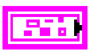


Inputs

i2c script reference in is a reference to an I^2C script that is run on an NI 845*x* device.

port number specifies the DIO port to write.

write value is the value to write to the DIO port. Only lines configured for output are updated.



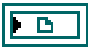
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

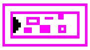
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



i2c script reference out is a reference to the I²C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

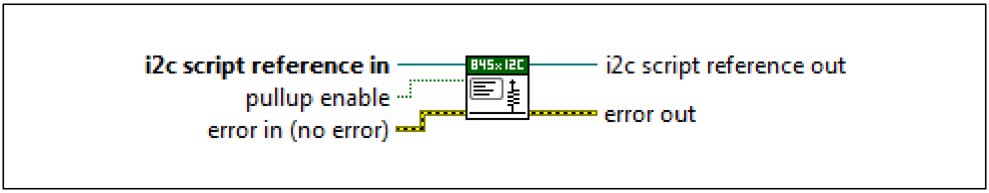
Description

Use **NI-845x I2C Script DIO Write Port.vi** to add an I²C Script DIO Write Port command to an I²C script referenced by **i2c script reference in**. This command allows you to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x I2C Script Pullup Enable.vi

Purpose

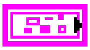
Adds an I²C Script Pullup Enable command to an I²C script referenced by **i2c script reference in**. This command enables or disables the internal pullups on an NI 845*x* device.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

pullup enable controls the enabled state of the internal pullups.



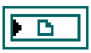
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

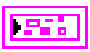
code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

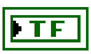
Outputs



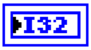
i2c script reference out is a reference to the I²C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

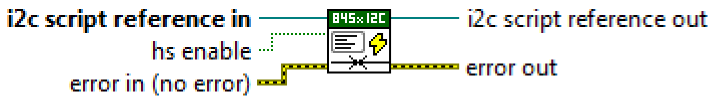
Description

Use **NI-845x I2C Script Pullup Enable.vi** to add an I^{2}C Script Pullup Enable command to an I^{2}C script referenced by **i2c script reference in**. Use this command to set the status of onboard pullups for I^{2}C operations. The pullup resistors pull SDA and SCL up to *I/O Voltage Level*.

NI-845x I2C Script HS Enable.vi

Purpose

Adds an I²C Script HS Enable command to an I²C script referenced by **i2c script reference in**. This command enables or disables High Speed mode on an NI 845*x* device.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

hs enable sets the High Speed mode to enabled or disabled on an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

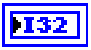
source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I²C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

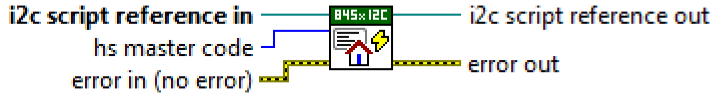
Use **NI 845x I2C Script HS Enable.vi** to add an I^2C Script HS Enable command to an I^2C script referenced by **i2c script reference in**. Use this command to enable High Speed mode. High Speed mode must be enabled to use the High Speed clock rate or the High Speed master code.

High Speed mode is described in the NXP I^2C Specification.

NI-845x I2C Script HS Master Code.vi

Purpose

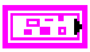
Adds an I²C Script HS Master Code command to an I²C script referenced by **i2c script reference in**. This command transfers the master code for High Speed mode on an NI 845*x* device.



Inputs

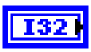
i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

hs master code sets the lower 3 bits of the master code on the NI 845*x* device.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

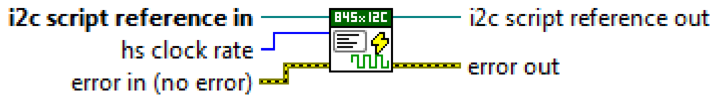
Description

Use **NI 845x I2C HS Master Code.vi** to add an I^{2}C Script HS Master Code command to an I^{2}C script referenced by **i2c script reference in**. This command writes a master code to the I^{2}C bus connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. This command assumes that a start condition previously has been issued to the I^{2}C bus using an I^{2}C script start command. The master code is internally set to 00001XXX. The lower three bits are set using the I^{2}C Script HS Master Code command. After the master code is transferred, the device waits for the slave device on the I^{2}C bus to acknowledge or not acknowledge the master code. If a slave acknowledges the master code, **NI-845x I2C Run Script.vi** exits with an error.

NI-845x I2C Script HS Clock Rate.vi

Purpose

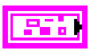
Adds an I²C Script HS Clock Rate command to an I²C script referenced by **i2c script reference in**. This command sets the I²C High Speed clock rate.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

hs clock rate specifies the I²C High Speed clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which clock rates your NI 845*x* device supports.



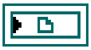
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

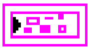
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



i2c script reference out is a reference to the I²C script after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

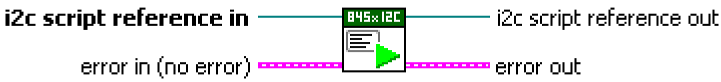
Description

Use **NI-845x I2C Script HS Clock Rate.vi** to add an I²C Script High Speed Clock Rate command to an I²C script referenced by **i2c script reference in**. This command sets the I²C High Speed clock rate for the I²C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845*x* driver adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

NI-845x I2C Script Issue Start.vi

Purpose

Adds an I^{2}C Script Issue Start command to an I^{2}C script referenced by **i2c script reference in**. This command issues a start condition on the I^{2}C bus.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute

the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

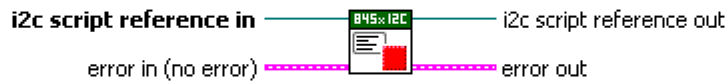
Description

Use **NI-845x I2C Script Issue Start.vi** to add an I^{2}C Script Issue Start command to an I^{2}C script referenced by **i2c script reference in**. This command issues a start condition on the I^{2}C bus connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. This command first waits for the I^{2}C bus to be free. If the I^{2}C bus is not free within the one second timeout of your NI 845*x* device, an error is returned when **NI-845x I2C Run Script.vi** is executed. If the bus is free before the timeout, the NI 845*x* device issues the start condition on the I^{2}C bus connected to the specified I^{2}C port. This command should also be used to issue a restart condition within an I^{2}C transaction.

NI-845x I2C Script Issue Stop.vi

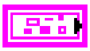
Purpose

Adds an I²C Script Issue Stop command to an I²C script referenced by **i2c script reference in**. This command issues a stop condition on the I²C bus.



Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.



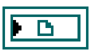
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

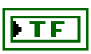
source identifies the VI where the error occurred.

Outputs

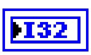


i2c script reference out is a reference to the I²C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute

the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

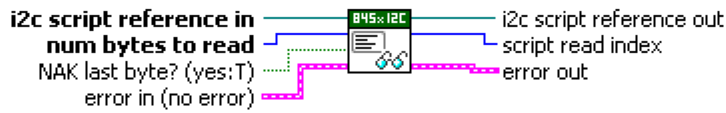
Description

Use **NI-845x I2C Script Issue Stop.vi** to add an I^{2}C Script Issue Stop command to an I^{2}C script referenced by **i2c script reference in**. This command issues a stop condition on the I^{2}C bus connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. Per the NXP I^{2}C Specification, all I^{2}C transactions must be terminated with a stop condition.

NI-845x I2C Script Read.vi

Purpose

Adds an I²C Script Read command to an I²C script referenced by **i2c script reference in**. This command reads an array of data from an I²C slave device.

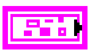


Inputs

i2c script reference in is a reference to an I²C script that is run on an NI 845*x* device.

num bytes to read specifies the number of bytes to read from an I²C slave.

NAK Last Byte? sets whether the last byte read is acknowledged (FALSE) or not acknowledged (TRUE) by the I²C interface. If **NAK Last Byte?** is TRUE, all bytes up to the last byte read are acknowledged. The last byte read is not acknowledged. If **NAK Last Byte?** is FALSE, all bytes are acknowledged.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

i2c script reference out is a reference to the I^{2}C script after this VI runs.

script read index is the index of the read command within the script. It is used as an input into **NI-845x I2C Extract Script Read Data.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

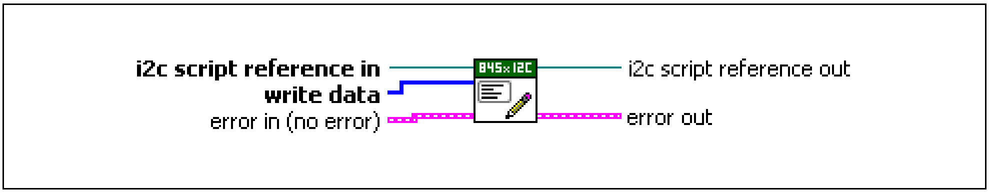
Use **NI-845x I2C Script Read.vi** to add an I^{2}C Script Read command to an I^{2}C script referenced by **i2c script reference in**. This command reads an array of data from a device connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. This command assumes that a start condition and address+read condition have been issued to the I^{2}C bus using prior I^{2}C script commands. It clocks in **num bytes to read** bytes from the I^{2}C slave device, acknowledging each byte up to the last one. Depending on the type of I^{2}C transaction you want to build, you may want to acknowledge (ACK) or not acknowledge (NAK) the last data byte read, which you can specify with the **NAK last byte?** input.

To obtain the data read from the specified I^{2}C port, you can wire **script read index** to **NI-845x I2C Extract Script Read Data.vi** after execution of the script, which returns the data read by this script command.

NI-845x I2C Script Write.vi

Purpose

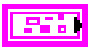
Adds an I^{2}C Script Write command to an I^{2}C script referenced by **i2c script reference in**. This command writes an array of data to an I^{2}C slave device.



Inputs

i2c script reference in is a reference to an I^{2}C script that is run on an NI 845*x* device.

write data contains an array of data to write to the I^{2}C slave.



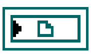
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

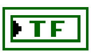
source identifies the VI where the error occurred.

Outputs

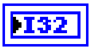


i2c script reference out is a reference to the I^{2}C script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x I2C Script Write.vi** to add an I^{2}C Script Write command to an I^{2}C script referenced by **i2c script reference in**. This command writes an array of data to an I^{2}C slave device connected to the I^{2}C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. This command assumes that a start condition and address+write condition have been issued to the I^{2}C bus using prior I^{2}C script commands. It clocks the **write data** array into the I^{2}C slave device, testing for a slave device acknowledge after transmission of each byte. If a slave does not acknowledge a byte, **NI-845x I2C Run Script.vi** exits with an error.

7

NI-845*x* I^{2}C API for C

This chapter lists the functions for the NI-845*x* I^{2}C API and describes the format, purpose, and parameters for each function. The functions are listed alphabetically in four categories: general device, configuration, basic, and advanced.

Section Headings

The NI-845*x* I^{2}C API for C functions include the following section headings.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Inputs and Outputs

These sections list the function input and output parameters.

Description

The description section gives details about the purpose and effect of each function.

Data Types

The NI-845*x* I^{2}C API for C functions use the following data types.

Data Type	Purpose
`uInt8`	8-bit unsigned integer
`uInt16`	16-bit unsigned integer
`uInt32`	32-bit unsigned integer
`int8`	8-bit signed integer

Data Type	Purpose
`int16`	16-bit signed integer
`int32`	32-bit signed integer
`uInt8 *`	Pointer to an 8-bit unsigned integer
`uInt16 *`	Pointer to a 16-bit unsigned integer
`uInt32 *`	Pointer to a 32-bit unsigned integer
`int8 *`	Pointer to an 8-bit signed integer
`int16 *`	Pointer to a 16-bit signed integer
`int32 *`	Pointer to a 32-bit signed integer
`char *`	ASCII string represented as an array of characters terminated by null character (`'\0'`)

List of Functions

The following table contains an alphabetical list of the NI-845*x* I²C API for C functions.

Function	Purpose
`ni845xClose`	Closes a previously opened NI 845*x* device.
`ni845xCloseFindDeviceHandle`	Closes the handles created by `ni845xFindDevice`.
`ni845xDeviceLock`	Locks NI 845*x* devices for access by a single thread.
`ni845xDeviceUnlock`	Unlocks NI 845*x* devices.
`ni845xFindDevice`	Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.
`ni845xFindDeviceNext`	Finds subsequent devices after `ni845xFindDevice` has been called.
`ni845xI2cConfigurationClose`	Closes an NI-845*x* I²C I/O configuration.
`ni845xI2cConfigurationGetAddress`	Retrieves the configuration's address.

Function	Purpose
ni845xI2cConfigurationGetAddressSize	Retrieves the configuration's address size.
ni845xI2cConfigurationGetClockRate	Retrieves the configuration's clock rate in kilohertz.
ni845xI2cConfigurationGetHSClockRate	Retrieves the configuration's High Speed clock rate in kilohertz.
ni845xI2cConfigurationGetHSEnable	Retrieves the configuration's High Speed enable setting.
ni845xI2cConfigurationGetHSMasterCode	Retrieves the configuration's High Speed master code.
ni845xI2cConfigurationGetPort	Retrieves the configuration's port value.
ni845xI2cConfigurationOpen	Creates a new NI-845x I²C configuration.
ni845xI2cConfigurationSetAddress	Sets the configuration's address.
ni845xI2cConfigurationSetAddressSize	Sets the configuration's address size.
ni845xI2cConfigurationSetClockRate	Sets the configuration's clock rate in kilohertz.
ni845xI2cConfigurationSetHSClockRate	Sets the configuration's High Speed clock rate in kilohertz.
ni845xI2cConfigurationSetHSEnable	Sets the configuration's High Speed enable setting.
ni845xI2cConfigurationSetHSMasterCode	Sets the configuration's High Speed master code.
ni845xI2cConfigurationSetPort	Sets the configuration's port number.
ni845xI2cRead	Reads an array of data from an I²C slave device.
ni845xI2cScriptAddressRead	Adds an I²C Script Address+Read command to an I²C script referenced by ScriptHandle. This command writes a 7-bit address to the I²C bus. The direction bit is internally set to 1 for read.

Function	Purpose
`ni845xI2cScriptAddressWrite`	Adds an I²C Script Address+Write command to an I²C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I²C bus. The direction bit is internally set to 0 for write.
`ni845xI2cScriptClockRate`	Adds an I²C Script Clock Rate command to an I²C script referenced by `ScriptHandle`. This command sets the I²C clock rate.
`ni845xI2cScriptClose`	Closes an I²C script.
`ni845xI2cScriptDelay`	Adds an I²C Script Delay command to an I²C script referenced by `ScriptHandle`. This command adds a delay after the previous I²C script command.
`ni845xI2cScriptDioConfigureLine`	Adds an I²C Script DIO Configure Line command to an I²C script referenced by `ScriptHandle`. This command configures a DIO line on an NI 845*x* device.
`ni845xI2cScriptDioConfigurePort`	Adds an I²C Script DIO Configure Port command to an I²C script referenced by `ScriptHandle`. This command configures a DIO port on an NI 845*x* device.
`ni845xI2cScriptDioReadLine`	Adds an I²C Script DIO Read Line command to an I²C script referenced by `ScriptHandle`. This command reads from a DIO line on an NI 845*x* device.
`ni845xI2cScriptDioReadPort`	Adds an I²C Script DIO Read Port command to an I²C script referenced by `ScriptHandle`. This command reads from a DIO port on an NI 845*x* device.
`ni845xI2cScriptDioWriteLine`	Adds an I²C Script DIO Write Line command to an I²C script referenced by `ScriptHandle`. This command writes to a DIO line on an NI 845*x* device.

Function	Purpose
ni845xI2cScriptDioWritePort	Adds an I^{2}C Script DIO Write Port command to an I^{2}C script referenced by ScriptHandle. This command writes to a DIO port on an NI 845*x* device.
ni845xI2cScriptPullupEnable	Adds an I^{2}C Script Pullup Enable command to an I^{2}C script referenced by ScriptHandle. This command enables or disables the internal I^{2}C pullup resistors. The pullups connect to ni845xSetIoVoltageLevel.
ni845xI2cScriptExtractReadData	Extracts the desired read data from an I^{2}C script, referenced by ScriptHandle, which has been processed by ni845xI2cScriptRun. Each script read command (ni845xI2cScriptRead, ni845xI2cScriptDioReadPort, ni845xI2cScriptDioReadLine) returns a script read index. You can extract data for each script read index in a script, by passing each index to ni845xI2cScriptExtractReadData.
ni845xI2cScriptExtractReadDataSize	Retrieves the read data size from an I^{2}C script, referenced by ScriptHandle, which has been processed by ni845xI2cScriptRun. Each script read command (ni845xI2cScriptRead, ni845xI2cScriptDioReadPort, ni845xI2cScriptDioReadLine) returns a script read index. You can extract data for each script read index in a script, by passing each index to ni845xI2cScriptExtractReadData.
ni845xI2cScriptHSEnable	Adds an I^{2}C Script HS Enable command to an I^{2}C script referenced by ScriptHandle. This command enables the I^{2}C port to run in high-speed mode.

Function	Purpose
ni845xI2cScriptHSMasterCode	Adds an I^{2}C Script HS Master Code command to an I^{2}C script referenced by ScriptHandle. This command configures the I^{2}C master code, which is used to initiate High Speed I^{2}C mode.
ni845xI2cScriptHSClockRate	Adds an I^{2}C Script HS Clock Rate command to an I^{2}C script referenced by ScriptHandle. This command sets the High Speed I^{2}C clock rate.
ni845xI2cScriptIssueStart	Adds an I^{2}C Script Issue Start command to an I^{2}C script indicated by ScriptHandle. This command issues a start condition on the I^{2}C bus.
ni845xI2cScriptIssueStop	Adds an I^{2}C Script Issue Stop command to an I^{2}C script referenced by ScriptHandle. This command issues a stop condition on the I^{2}C bus.
ni845xI2cScriptOpen	Opens an empty I^{2}C script to begin adding commands to.
ni845xI2cScriptRead	Adds an I^{2}C Script Read command to an I^{2}C script referenced by ScriptHandle. This command reads an array of data from an I^{2}C slave device.
ni845xI2cScriptReset	Resets an I^{2}C script referenced by ScriptHandle to an empty state.
ni845xI2cScriptRun	Sends the I^{2}C script to the desired NI 845*x* device, which then interprets and runs it.
ni845xI2cScriptWrite	Adds an I^{2}C Script Write command to an I^{2}C script referenced by ScriptHandle. This command writes an array of data to an I^{2}C slave device.
ni845xI2cSetPullupEnable	Enables or disables the onboard I^{2}C pullups.
ni845xI2cWrite	Writes an array of data to an I^{2}C slave device.
ni845xI2cWriteRead	Performs a write followed by read (combined format) on an I^{2}C slave device.

Function	Purpose
ni845xOpen	Opens an NI 845*x* device for use with various write, read, and device property functions.
ni845xSetIoVoltageLevel	Sets the voltage level of the NI-845*x* I/O pins (DIO/SPI/VioRef).
ni845xStatusToString	Converts a status code into a descriptive string.

General Device

ni845xClose

Purpose

Closes a previously opened NI 845*x* device.

Format

```
int32 ni845xClose(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be closed.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xClose` to close a device handle previously opened by `ni845xOpen`. Passing an invalid handle to `ni845xClose` is ignored.

ni845xCloseFindDeviceHandle

Purpose

Closes the handles created by ni845xFindDevice.

Format

```
int32 ni845xCloseFindDeviceHandle (
      uInt32 FindDeviceHandle
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. ni845xFindDevice creates this parameter.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to ni845xStatusToString.

Description

Use `ni845xCloseFindDeviceHandle` to close a find list. In this process, all allocated data structures are freed.

ni845xDeviceLock

Purpose

Locks NI 845*x* devices for access by a single thread.

Format

```
int32 ni845xDeviceLock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be locked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

This function locks NI 845*x* devices and prevents multiple processes or threads from accessing the device until the process or thread that owns the device lock calls an equal number of `ni845xDeviceUnlock` calls. Any thread or process that attempts to call `ni845xDeviceLock` when the device is already locked is forced to sleep by the operating system. This is useful for when multiple Basic API device accesses must occur uninterrupted by any other processes or threads. If a thread exits without fully unlocking the device, the device is unlocked. If a thread is the current owner of the lock, and calls `ni845xDeviceLock` again, the thread will not deadlock itself, but care must be taken to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` called. This function can possibly lock a device indefinitely: If a thread never calls `ni845xDeviceUnlock`, or fails to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` call, and never exits, other processes and threads are forced to wait. This is *not* recommended for users unfamiliar with threads or processes. A simpler alternative is to use scripts. Scripts provide the same capability to ensure transfers are uninterrupted, and with possible performance benefits.

ni845xDeviceUnlock

Purpose

Unlocks NI 845*x* devices.

Format

```
int32 ni845xDeviceUnlock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be unlocked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDeviceUnlock` to unlock access to an NI 845*x* device previously locked with `ni845xDeviceLock`. Every call to `ni845xDeviceLock` must have a corresponding call to `ni845xDeviceUnlock`. Refer to `ni845xDeviceLock` for more details regarding how to use device locks.

ni845xFindDevice

Purpose

Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.

Format

```
int32 ni845xFindDevice (
      char   * pFirstDevice,
      uInt32 * pFindDeviceHandle,
      uInt32 * pNumberFound
      );
```

Inputs

None.

Outputs

`char * pFirstDevice`

A pointer to the string containing the first NI 845*x* device found. You can pass this name to the `ni845xOpen` function to open the device. If no devices exist, this is an empty string.

`uInt32 * pFindDeviceHandle`

Returns a handle identifying this search session. This handle is used as an input in `ni845xFindDeviceNext` and `ni845xCloseFindDeviceHandle`.

`uInt32 * pNumberFound`

A pointer to the total number of NI 845*x* devices found in the system. You can use this number in conjunction with the `ni845xFindDeviceNext` function to find a particular device. If no devices exist, this returns 0.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDevice` to get a single NI 845*x* device and the number of NI 845*x* devices in the system. You can then pass the string returned to `ni845xOpen` to access the device. If you must discover more devices, use `ni845xFindDeviceNext` with `pFindDeviceHandle` and pNumberFound to find the remaining NI 845*x* devices in the system. After finding all desired devices, call ni845xCloseFindDeviceHandle to close the device handle and relinquish allocated resources.

Note pFirstDevice must be at least 256 bytes.

Note pFindDeviceHandle and pNumberFound are optional parameters. If only the first match is important, and the total number of matches is not needed, you can pass in a NULL pointer for both of these parameters, and the NI-845*x* driver automatically calls ni845xCloseFindDeviceHandle before this function returns.

ni845xFindDeviceNext

Purpose

Finds subsequent devices after `ni845xFindDevice` has been called.

Format

```
int32 ni845xFindDeviceNext (
      uInt32 FindDeviceHandle,
      char * pNextDevice
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

`char * pNextDevice`

A pointer to the string containing the next NI 845*x* device found. This is empty if no further devices are left.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDeviceNext` after first calling `ni845xFindDevice` to find the remaining devices in the system. You can then pass the string returned to `ni845xOpen` to access the device.

Note `pNextDevice` must be at least 256 bytes.

ni845xOpen

Purpose

Opens an NI 845*x* device for use with various write, read, and device property functions.

Format

```
int32 ni845xOpen (
       char   * pResourceName,
       uInt32 * pDeviceHandle
       );
```

Inputs

`char * pResourceName`

A resource name string corresponding to the NI 845*x* device to be opened.

Outputs

`uInt32 * pDeviceHandle`

A pointer to the device handle.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xOpen` to open an NI 845*x* device for access. The string passed to `ni845xOpen` can be any of the following: an `ni845xFindDevice` device string, an `ni845xFindDeviceNext` device string, a Measurement & Automation Explorer resource name, or a Measurement & Automation Explorer alias.

ni845xSetIoVoltageLevel

Purpose

Modifies the voltage output from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSetIoVoltageLevel (
       uInt32  DeviceHandle,
       uInt8   VoltageLevel
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 VoltageLevel`

The desired voltage level. `VoltageLevel` uses the following values:

- `kNi845x33Volts` (33): The output I/O high level is 3.3 V.
- `kNi845x25Volts` (25): The output I/O high level is 2.5 V.
- `kNi845x18Volts` (18): The output I/O high level is 1.8 V.
- `kNi845x15Volts` (15): The output I/O high level is 1.5 V.
- `kNi845x12Volts` (12): The output I/O high level is 1.2 V.

The default value of this property is 3.3 V.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSetIoVoltageLevel` to modify the board reference voltage of the NI 845*x* device. The board reference voltage is used for SPI, I²C, and DIO. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the available voltage levels on your hardware.

ni845xI2cSetPullupEnable

Purpose

Modifies the voltage output from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xI2cSetPullupEnable (
      uInt32 DeviceHandle,
      uInt8 Enable
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 Enable`

The setting for the pullup resistors. `Enable` uses the following values:

- `kNi845xPullupDisable` (0): Pullups are disabled.
- `kNi845xPullupEnable` (1): Pullups are enabled.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cPullupEnable` to enable or disable the onboard pullup resistors for I²C operations. The pullup resistors pull SDA and SCL up to `ni845xSetIoVoltageLevel`.

ni845xStatusToString

Purpose

Converts a status code into a descriptive string.

Format

```
void ni845xStatusToString (
       int32  StatusCode,
       uInt32 MaxSize,
       int8 * pStatusString
       );
```

Inputs

`int32 StatusCode`

Status code returned from an NI-845*x* function.

`uInt32 MaxSize`

Size of the `pStatusString` buffer (in bytes).

Outputs

`int8 * pStatusString`

ASCII string that describes `StatusCode`.

Description

When the status code returned from an NI-845*x* function is nonzero, an error or warning is indicated. This function obtains a description of the error/warning for debugging purposes.

The return code is passed into the `StatusCode` parameter. The `MaxSize` parameter indicates the number of bytes available in `pStatusString` for the description (including the NULL character). The description is truncated to size `MaxSize` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `String` is null-terminated, so you can use it with ANSI C functions such as `printf`.

For applications written in C or C++, each NI-845*x* function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845*x* use of this status.

NI-845*x* Status Codes

Status Code	Meaning
Negative	Error—Function did not perform expected behavior.
Positive	Warning—Function executed, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every NI-845*x* function. If an error is detected, you should close all NI-845*x* handles, then exit the application. If a warning is detected, you can display a message for debugging purposes, or simply ignore the warning.

In some situations, you may want to check for specific errors in the code and continue communication when they occur. For example, when communicating to an I^2C EEPROM, you may expect the device to NAK its address during a write cycle, and you may use this knowledge to poll for when the write cycle has completed.

Configuration

ni845xI2cConfigurationClose

Purpose

Closes an I²C I/O configuration.

Format

```
int32 ni845xI2cConfigurationClose (
       uInt32 ConfigurationHandle
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationClose` to close a configuration.

ni845xI2cConfigurationGetAddress

Purpose

Retrieves the configuration address.

Format

```
int32 ni845xI2cConfigurationGetAddress (
      uInt32   ConfigurationHandle,
      uInt16 * pAddress
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt16 * pAddress`

A pointer to an unsigned 16-bit integer to store the I^2C slave address in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetAddress` to retrieve the I^2C configuration slave address as a 7-bit number.

ni845xI2cConfigurationGetAddressSize

Purpose

Retrieves the configuration address size.

Format

```
int32 ni845xI2cConfigurationGetAddressSize (
       uInt32  ConfigurationHandle,
       int32 * pSize
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`int32 * pSize`

A pointer to an unsigned 32-bit integer to store the address size in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetAddressSize` to retrieve the addressing scheme to use when addressing the I^2C slave device this configuration describes.

ni845xI2cConfigurationGetClockRate

Purpose

Retrieves the configuration clock rate in kilohertz.

Format

```
int32 ni845xI2cConfigurationGetClockRate (
      uInt32   ConfigurationHandle,
      uInt16 * pClockRate
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt16 * pClockRate`

A pointer to an unsigned 16-bit integer to store the clock rate in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetClockRate` to retrieve the I^{2}C clock rate in kilohertz. This retrieves the value currently stored in memory, which may not be compatible with your NI 845*x* device.

ni845xI2cConfigurationGetHSClockRate

Purpose

Retrieves the configuration High Speed clock rate in kilohertz.

Format

```
int32 ni845xI2cConfigurationGetHSClockRate (
       uInt32   ConfigurationHandle,
       uInt16 * pHSClockRate
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt16 * pHSClockRate`

A pointer to an unsigned 16-bit integer to store the clock rate in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetHSClockRate` to retrieve the I²C High Speed clock rate in kilohertz. This retrieves the value currently stored in memory, which may not be compatible with your NI 845*x* device.

ni845xI2cConfigurationGetHSEnable

Purpose

Retrieves the configuration High Speed enable status.

Format

```
int32 ni845xI2cConfigurationGetHSEnable (
      uInt32   ConfigurationHandle,
      uInt16 * pHSEnable
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt8 * pHSEnable`

A pointer to an unsigned 8-bit integer to store the enabled status in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetHSEnable` to retrieve the configuration High Speed enable status. This retrieves the value currently stored in memory, which may not be compatible with your NI 845*x* device.

ni845xI2cConfigurationGetHSMasterCode

Purpose

Retrieves the configuration master code.

Format

```
int32 ni845xI2cConfigurationGetHSMasterCode (
       uInt32  ConfigurationHandle,
       uInt8 * pHSMasterCode
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt16 * pHSMasterCode`

A pointer to an unsigned 8-bit integer to store the master code in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetHSMasterCode` to retrieve the I^{2}C High Speed master code. This retrieves the value currently stored in memory, which may not be compatible with your NI 845*x* device.

ni845xI2cConfigurationGetPort

Purpose

Retrieves the configuration port value.

Format

```
int32 ni845xI2cConfigurationGetPort (
       uInt32 ConfigurationHandle,
       uInt8 * pPort
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

Outputs

`uInt8 * pPort`

A pointer to an unsigned byte to store the port value in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationGetPort` to retrieve the I^2C port that this configuration communicates across.

ni845xI2cConfigurationOpen

Purpose

Creates a new NI-845*x* I^2C configuration.

Format

```
int32 ni845xI2cConfigurationOpen (
      uInt32 * pConfigurationHandle
      );
```

Inputs

None.

Outputs

`uInt32 * pConfigurationHandle`

A pointer to an unsigned 32-bit integer to store the configuration handle in. This must not be NULL.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use this function to create a new configuration to use with the NI-845*x* I^2C Basic API. Pass the handles to the `ni845xI2cConfigurationSet*` series of functions to modify the configuration properties. Then, pass the configuration to the I^2C basic functions to execute them on the described I^2C slave. After you finish communicating with your I^2C slave, pass the handle to the `ni845xI2cConfigurationSet*` series of functions to reconfigure it or use `ni845xI2cConfigurationClose` to delete the configuration.

ni845xI2cConfigurationSetAddress

Purpose

Sets the configuration address.

Format

```
int32 ni845xI2cConfigurationSetAddress (
      uInt32 ConfigurationHandle,
      uInt16 Address
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt16 Address`

The slave address. For 7-bit device addressing, the NXP I²C specification defines a 7-bit slave address and a direction bit. During the address phase of an I²C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845*x* software follows the convention used in the NXP I²C specification and defines an address for a 7-bit device as a 7-bit value. The NI-845*x* software internally sets the direction bit to the correct value, depending on the function (write or read). Some manufacturers specify the address for their 7-bit device as a byte. In such cases, bits 7–1 contain the slave address, and bit 0 contains the direction. When using the NI-845*x* software, discard the direction bit and right-shift the byte value by one to create the 7-bit address.

The address default value is 0.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetAddress` to set the I²C slave address. This is a 7-bit number; do not include the direction bit.

ni845xI2cConfigurationSetAddressSize

Purpose

Sets the configuration address size.

Format

```
int32 ni845xI2cConfigurationSetAddressSize (
      uInt32 ConfigurationHandle,
      int32  Size
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`int32 Size`

The addressing scheme to use when addressing the I²C slave device this configuration describes. `Size` uses the following values:

- `kNi845xI2cAddress7Bit` (0): The NI 845*x* hardware uses the standard 7-bit addressing when communicating with the I²C slave device.
- `kNi845xI2cAddress10Bit` (1): The NI 845*x* hardware uses the extended 10-bit addressing when communicating with the I²C slave device.

The address default value is `kNi845xI2cAddress7Bit`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetAddressSize` to set the configuration address size as either 7 bits or 10 bits.

ni845xI2cConfigurationSetClockRate

Purpose

Sets the configuration clock rate in kilohertz.

Format

```
int32 ni845xI2cConfigurationSetClockRate (
      uInt32 ConfigurationHandle,
      uInt16 ClockRate
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt16 ClockRate`

Specifies the I²C clock rate in kilohertz. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports. If your hardware does not support the supplied clock rate, a warning is generated, and the next smallest supported clock rate is used. If the supplied clock rate is smaller than the smallest supported clock rate, an error is generated.

The clock rate default value is 100 kHz.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetClockRate` to set the I²C configuration clock rate in kilohertz.

ni845xI2cConfigurationSetHSClockRate

Purpose

Sets the configuration High Speed clock rate in kilohertz.

Format

```
int32 ni845xI2cConfigurationSetHSClockRate (
       uInt32 ConfigurationHandle,
       uInt16 HSClockRate
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt16 HSClockRate`

Specifies the I²C clock rate in kilohertz. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which High Speed clock rates your NI 845*x* device supports. If your hardware does not support the supplied clock rate, a warning is generated, and the next smallest supported clock rate is used. If the supplied clock rate is smaller than the smallest supported clock rate, an error is generated.

The clock rate default value is 1666 Hz.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetHSClockRate` to set the I²C configuration High Speed clock rate in kilohertz.

ni845xI2cConfigurationSetHSEnable

Purpose

Sets the configuration High Speed enabled status.

Format

```
int32 ni845xI2cConfigurationSetHSEnable (
      uInt32 ConfigurationHandle,
      uInt8 HSEnable
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt8 HSEnable`

Specifies the I²C High Speed enabled status. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine if your NI 845*x* device supports I²C High Speed mode. If your hardware does not support I²C High Speed Mode, an error is generated. `HSEnable` uses the following values:

- `kNi845xHSDisable` (0): Disable High Speed mode.
- `kNi845xHSEnable` (1): Enable High Speed mode.

The default value is `kNi845xHSDisable`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetHSEnable` to set the I²C High Speed enabled status.

ni845xI2cConfigurationSetHSMasterCode

Purpose

Sets the configuration High Speed master code.

Format

```
int32 ni845xI2cConfigurationSetHSMasterCode (
       uInt32 ConfigurationHandle,
       uInt8 HSMasterCode
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt8 HSMasterCode`

Specifies the I^2C High Speed master code.

The default value is 1.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetHSMasterCode` to set the I^2C configuration High Speed master code.

ni845xI2cConfigurationSetPort

Purpose

Sets the configuration port number.

Format

```
int32 ni845xI2cConfigurationSetPort (
       uInt32 ConfigurationHandle,
       uInt8  PortNumber
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt8 Port`

Specifies the I^{2}C port that this configuration communicates across.

Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the number of I^{2}C ports your NI 845*x* device supports.

The port number default value is 0.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cConfigurationSetPort` to select the port where the I^{2}C slave device resides.

Basic

ni845xI2cRead

Purpose

Reads an array of data from an I²C slave device.

Format

```
int32 ni845xI2cRead (
       uInt32   DeviceHandle,
       uInt32   ConfigurationHandle,
       uInt32   NumBytesToRead,
       uInt32 * pReadSize,
       uInt8  * pReadData
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

Configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt32 NumBytesToRead`

The number of bytes to read. This must be nonzero.

Outputs

`uInt32 * pReadSize`

A pointer to the amount of bytes read.

`uInt8 * pReadData`

A pointer to an array of bytes where the bytes that have been read are stored.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cRead` to read an array of data from an I^{2}C slave device. Per the NXP I^{2}C specification, each byte read up to the last byte is acknowledged. The last byte is not acknowledged. This function first waits for the I^{2}C bus to be free. If the I^{2}C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7-bit or 10-bit I^{2}C read transaction, per the NXP I^{2}C specification. The address type (7-bit or 10-bit) and other configuration parameters are specified by `ConfigurationHandle`. If the NI 845*x* device tries to access the bus at the same time as another I^{2}C master device and loses arbitration, the read transaction is terminated and an error is returned. If the slave device does not acknowledge the transaction address, an error is returned. Otherwise, the transaction is completed, and a stop condition is generated per the NXP I^{2}C specification.

Before using `ni845xI2cRead`, you must ensure that the configuration parameters specified in `ConfigurationHandle` are correct for the device you want to access.

ni845xI2cWrite

Purpose

Writes an array of data to an I^2C slave device.

Format

```
int32 ni845xI2cWrite (
      uInt32  DeviceHandle,
      uInt32  ConfigurationHandle,
      uInt32  WriteSize,
      uInt8 * pWriteData
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

Configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt32 WriteSize`

The number of bytes to write. This must be nonzero.

`uInt8 * pWriteData`

A pointer to an array of bytes where the data to be written resides.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cWrite` to write an array of data to an I^2C slave device. This function first waits for the I^2C bus to be free. If the I^2C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7-bit or 10-bit I^2C write transaction, per the NXP I^2C specification. The address type (7-bit or 10-bit) and other configuration parameters are specified by `ConfigurationHandle`. If the NI 845*x* device tries to access the bus at the same time as another I^2C master device and loses arbitration, the write transaction is terminated and an error is returned. If the slave device does not acknowledge any transaction byte, an error is returned. Otherwise, the transaction is completed, and a stop condition is generated per the NXP I²C specification.

ni845xI2cWriteRead

Purpose

Performs a write followed by read (combined format) on an I²C slave device.

Format

```
int32 ni845xI2cWriteRead (
       uInt32   DeviceHandle,
       uInt32   ConfigurationHandle,
       uInt32   WriteSize,
       uInt8  * pWriteData,
       uInt32   NumBytesToRead,
       uInt32 * pReadSize,
       uInt8  * pReadData
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

Configuration handle returned from `ni845xI2cConfigurationOpen`.

`uInt32 WriteSize`

The number of bytes to write. This must be nonzero.

`uInt8 * pWriteData`

A pointer to an array of bytes where the data to be written resides.

`uInt32 NumBytesToRead`

An unsigned 32-bit integer corresponding to the number of bytes to read. This must be nonzero.

Outputs

`uInt32 * pReadSize`

A pointer to the amount of bytes read.

`uInt8 * pReadData`

A pointer to an array of bytes where the bytes that have been read are stored.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cWriteRead` to perform a write followed by read (combined format) on an I²C slave device. During the transaction read portion, per the NXP I²C specification, each byte read up to the last byte is acknowledged. The last byte is not acknowledged. This function first waits for the I²C bus to be free. If the I²C bus is not free within the one second timeout of your NI 845*x* device, an error is returned. If the bus is free before the timeout, the NI 845*x* device executes a 7-bit or 10-bit I²C write/read transaction. Per the NXP I²C specification, the write/read transaction consists of a start-write-restart-read-stop sequence. The address type (7-bit or 10-bit) and other configuration parameters are specified by `ConfigurationHandle`. If the NI 845*x* device tries to access the bus at the same time as another I²C master device and loses arbitration, the read transaction is terminated and an error is returned. If the slave device does not acknowledge an address or byte write within the transaction, an error is returned. Otherwise, the transaction is completed and a stop condition is generated per the NXP I²C specification. Note that this type of combined transaction is provided because it is commonly used (for example, with EEPROMs). The NXP I²C specification provides flexibility in the construction of I²C transactions. The NI-845*x* I²C scripting functions allow creating and customizing complex I²C transactions as needed.

Before using `ni845xI2cWriteRead`, you must ensure that the configuration parameters specified in `ConfigurationHandle` are correct for the device you want to access.

Advanced

ni845xI2cScriptAddressRead

Purpose

Adds an I²C Script Address+Read command to an I²C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I²C bus. The direction bit is internally set to 1 for read.

Format

```
int32 ni845xI2cScriptAddressRead (
       uInt32 ScriptHandle,
       uInt8  Address
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 Address`

The 7-bit slave address to read from.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptAddressRead` to add an I²C Script Address+Read command to an I²C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I²C bus connected to the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. The direction bit is internally set to 1 for read. This command assumes that a start condition has been previously issued to the I²C bus using an I²C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I²C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, `ni845xI2cScriptRun` exits with an error.

ni845xI2cScriptAddressWrite

Purpose

Adds an I^{2}C Script Address+Write command to an I^{2}C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I^{2}C bus. The direction bit is internally set to 0 for write.

Format

```
int32 ni845xI2cScriptAddressWrite (
       uInt32 ScriptHandle,
       uInt8  Address
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 Address`

The 7-bit I^{2}C slave address to write to.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptAddressWrite` to add an I^{2}C Script Address+Write command to an I^{2}C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I^{2}C bus connected to the I^{2}C port you specify when you use `ni845xI2cScriptRun` to execute the script. The direction bit is internally set to 0 for write. This command assumes that a start condition has been previously issued to the I^{2}C bus using an I^{2}C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I^{2}C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, `ni845xI2cScriptRun` exits with an error.

ni845xI2cScriptClockRate

Purpose

Adds an I²C Script Clock Rate command to an I²C script referenced by `ScriptHandle`. This command sets the I²C clock rate.

Format

```
int32 ni845xI2cScriptClockRate (
      uInt32 ScriptHandle,
      uInt16 ClockRate
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt16 ClockRate`

The I²C clock rate in kilohertz. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptClockRate` to add an I²C Script Clock Rate command to an I²C script referenced by `ScriptHandle`. This command sets the I²C clock rate for the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845x driver adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

ni845xI2cScriptClose

Purpose

Closes an I²C script.

Format

```
int32 ni845xI2cScriptClose (uInt32 ScriptHandle);
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptClose` to delete a script from memory.

ni845xI2cScriptDelay

Purpose

Adds an I^{2}C Script Delay command to an I^{2}C script referenced by `ScriptHandle`. This command adds a delay after the previous I^{2}C script command.

Format

```
int32 ni845xI2cScriptDelay (
       uInt32 ScriptHandle,
       uInt8  Delay
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 Delay`

The desired delay in milliseconds.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDelay` to add an I^{2}C Script Delay command to an I^{2}C script referenced by `ScriptHandle`. This command adds a delay after the previous I^{2}C script command in milliseconds.

ni845xI2cScriptDioConfigureLine

Purpose

Adds an I²C Script DIO Configure Line command to an I²C script referenced by `ScriptHandle`. This command configures a DIO line on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioConfigureLine (
      uInt32 ScriptHandle,
      uInt8  PortNumber,
      uInt8  LineNumber,
      int32  ConfigurationValue
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to configure.

`int32 ConfigurationValue`

The line configuration. `ConfigurationValue` uses the following values:

- `kNi845xDioInput` (0): The line is configured for input.
- `kNi845xDioOutput` (1): The line is configured for output.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioConfigureLine` to add an I²C Script DIO Configure Line command to an I²C script referenced by `ScriptHandle`. This command allows you to configure one line, specified by `LineNumber`, of a byte-wide DIO port, as an input or output. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xI2cScriptDioConfigurePort

Purpose

Adds an I²C Script DIO Configure Port command to an I²C script referenced by `ScriptHandle`. This command configures a DIO port on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioConfigurePort (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  ConfigurationValue
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port to configure.

`uInt8 ConfigurationValue`

Bitmap that specifies the function of each individual line of a port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioConfigurePort` to add an I²C Script DIO Configure Port command to an I²C script referenced by `ScriptHandle`. Use this command to configure all eight lines of a byte-wide DIO port. Setting a bit to 1 configures the corresponding DIO port line for output. Setting a bit to 0 configures the corresponding port line for input. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the port to configure. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xI2cScriptDioReadLine

Purpose

Adds an I²C Script DIO Read Line command to an I²C script referenced by `ScriptHandle`. This command reads from a DIO line on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioReadLine (
       uInt32   ScriptHandle,
       uInt8    PortNumber,
       uInt8    LineNumber,
       uInt32*  pScriptReadIndex
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to read.

Outputs

`uInt32 * pScriptReadIndex`

An unsigned 32-bit integer pointer that stores the script read index. `pScriptReadIndex` is the read command index within the script. It is used as an input into `ni845xI2cScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioReadLine` to add an I²C Script DIO Read command to an I²C script referenced by `ScriptHandle`. Use this command to read one line, specified by `LineNumber`, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

To obtain the logic level read from the specified DIO port line, pass the value of `pScriptReadIndex` to `ni845xI2cScriptExtractReadDataSize` to retrieve the read data size and `ni845xI2cScriptExtractReadData` after script execution. `ni845xI2cScriptExtractReadData` returns either `kNi845xDioLogicLow` if logic level read on the specified line was low or `kNi845xDioLogicHigh` if the logic level read on the specified line was high.

ni845xI2cScriptDioReadPort

Purpose

Adds an I²C Script DIO Read Port command to an I²C script referenced by `ScriptHandle`. This command reads from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioReadPort (
       uInt32   ScriptHandle,
       uInt8    PortNumber,
       uInt32 * pScriptReadIndex
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port to read.

Outputs

`uInt32 * pScriptReadIndex`

An unsigned 32-bit integer pointer that stores the script read index. `pScriptReadIndex` is the read command index within the script. It is used as an input into `ni845xI2cScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioReadPort` to add an I²C Script DIO Read Port command to an I²C script referenced by `ScriptHandle`. Use this command to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

To obtain the data byte read from the specified DIO port, pass the value of `pScriptReadIndex` to `ni845xI2cScriptExtractReadDataSize` to retrieve the read data size and `ni845xI2cScriptExtractReadData` after script execution, which returns the data byte read by this script command.

ni845xI2cScriptDioWriteLine

Purpose

Adds an I²C Script DIO Write Line command to an I²C script referenced by `ScriptHandle`. This command writes to a DIO line on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioWriteLine (
       uInt32  ScriptHandle,
       uInt8   PortNumber,
       uInt8   LineNumber,
       int32   WriteData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to write.

`int32 WriteData`

The value to write to the line. `WriteData` uses the following values:

- `kNi845xDioLogicLow` (0): The line is set to the logic low state.
- `kNi845xDioLogicHigh` (1): The line is set to the logic high state.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioWriteLine` to add an I²C Script DIO Write Line command to an I²C script referenced by `ScriptHandle`. Use this command to write one line, specified by `LineNumber`, of a byte-wide DIO port. If `WriteData` is `kNi845xDioLogicHigh`, the specified line's output is driven to a high logic level. If `WriteData` is `kNi845xDioLogicLow`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xI2cScriptDioWritePort

Purpose

Adds an I²C Script DIO Write Port command to an I²C script referenced by `ScriptHandle`. This command writes to a DIO port on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptDioWritePort (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  WriteData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 PortNumber`

The DIO port to write.

`uInt8 WriteData`

The value to write to the DIO port. Only lines configured for output are updated.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptDioWritePort` to add an I²C Script DIO Write Port command to an I²C script referenced by `ScriptHandle`. Use this command to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xI2cScriptPullupEnable

Purpose

Adds an I²C Script Pullup Enable command to an I²C script referenced by `ScriptHandle`. This command enables or disables the onboard pullup resistors on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptPullupEnable (
       uInt32  ScriptHandle,
       uInt8   Enable
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 Enable`

The setting for the pullup resistors. `Enable` uses the following values:

- `kNi845xPullupDisable` (0): Pullups are disabled.
- `kNi845xPullupEnable` (1): Pullups are enabled.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptPullupEnable` to add an I²C Script Pullup Enable command to an I²C script referenced by `ScriptHandle`. Use this command to enable or disable the onboard pullup resistors for I²C operations. The pullup resistors pull SDA and SCL up to `ni845xSetIoVoltageLevel`.

ni845xI2cScriptExtractReadData

Purpose

Extracts the desired read data from an I²C script, referenced by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index. You can extract data for each script read index in a script, by passing each index to `ni845xI2cScriptExtractReadData`.

Format

```
int32 ni845xI2cScriptExtractReadData (
        uInt32  ScriptHandle,
        uInt32  ScriptReadIndex,
        uInt8 * pReadData
        );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt32 ScriptReadIndex`

The index within the script whose data should be extracted.

Outputs

`uInt8 * pReadData`

A pointer to store the data returned for the script command specified by `ScriptReadIndex`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptExtractReadData` to extract the desired read data from an I²C script, indicated by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each I²C script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index.

ni845xI2cScriptExtractReadDataSize

Purpose

Retrieves the read data size from an I2C script, referenced by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index. You can extract data for each script read index in a script, by passing each index to `ni845xI2cScriptExtractReadData`.

Format

```
int32 ni845xI2cScriptExtractReadDataSize (
      uInt32   ScriptHandle,
      uInt32   ScriptReadIndex,
      uInt32 * pReadDataSize
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt32 ScriptReadIndex`

The read in the script whose data should be extracted.

Outputs

`uInt32 * pReadDataSize`

Stores the read data buffer size at the given script read index.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptExtractReadDataSize` to retrieve the desired read data size from an I2C script, indicated by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each I2C script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index.

ni845xI2cScriptHSEnable

Purpose

Adds an I²C Script HS Enable command to an I²C script referenced by `ScriptHandle`. This command enables or disables High Speed mode on an NI 845*x* device.

Format

```
int32 ni845xI2cScriptHSEnable (
      uInt32 ScriptHandle,
      uInt8 HSEnable
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 HSEnable`

Enables or disables I²C High Speed mode. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your NI 845*x* device supports High Speed mode. `HSEnable` uses the following values:

- `kNi845xHSDisable` (0): Disable High Speed mode.
- `kNi845xHSEnable` (1): Enable High Speed mode.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptHSEnable` to add an I²C Script High Speed enable command to an I²C script referenced by `ScriptHandle`. This command sets the I²C High Speed enable status for the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. If the hardware does not support High Speed mode, the NI-845*x* driver generates an error.

ni845xI2cScriptHSMasterCode

Purpose

Adds an I²C Script High Speed Master Code command to an I²C script referenced by `ScriptHandle`. This command sets the I²C High Speed master code.

Format

```
int32 ni845xI2cScriptHSMasterCode (
      uInt32 ScriptHandle,
      uInt8 HSMasterCode
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt8 HSMasterCode`

The lower 3 bits of the I²C High Speed master code byte.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptHSMasterCode` to add an I²C Script HS Master Code command to an I²C script referenced by `ScriptHandle`. This command writes a master code to the I²C bus connected to the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. This command assumes a start condition previously has been issued to the I²C bus using an I²C script start command. The master code is internally set to 00001XXX. The lower three bits are set using `HSMasterCode`. After the master code is transferred, the device waits for slave device on the I²C bus to acknowledge or not acknowledge the master code. If a slave acknowledges the master code, `ni845xI2cScriptRun` exits with an error.

ni845xI2cScriptHSClockRate

Purpose

Adds an I²C Script High Speed Clock Rate command to an I²C script referenced by `ScriptHandle`. This command sets the I²C High Speed clock rate.

Format

```
int32 ni845xI2cScriptHSClockRate (
      uInt32 ScriptHandle,
      uInt8 HSClockRate
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt16 HSClockRate`

Specifies the I²C High Speed clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which High Speed clock rates your NI 845*x* device supports.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptHSClockRate` to add an I²C Script High Speed Clock Rate command to an I²C script referenced by `ScriptHandle`. This command sets the I²C High Speed clock rate for the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845*x* driver adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

ni845xI2cScriptIssueStart

Purpose

Adds an I²C Script Issue Start command to an I²C script indicated by `ScriptHandle`. This command issues a start condition on the I²C bus.

Format

```
int32 ni845xI2cScriptIssueStart (
       uInt32 ScriptHandle
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptIssueStart` to add an I²C Script Issue Start command to an I²C script referenced by `ScriptHandle`. This command issues a start condition on the I²C bus connected to the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. This command first waits for the I²C bus to be free. If the I²C bus is not free within the one second timeout of your NI 845*x* device, an error is returned when `ni845xI2cScriptRun` is executed. If the bus is free before the timeout, the NI 845*x* device issues the start condition on the I²C bus connected to the specified I²C port. This command should also be used to issue a restart condition within an I²C transaction.

ni845xI2cScriptIssueStop

Purpose

Adds an I²C Script Issue Stop command to an I²C script referenced by `ScriptHandle`. This command issues a stop condition on the I²C bus.

Format

```
int32 ni845xI2cScriptIssueStop (
      uInt32 ScriptHandle
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptIssueStop` to add an I²C Script Issue Stop command to an I²C script referenced by `ScriptHandle`. This command issues a stop condition on the I²C bus connected to the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. Per the NXP I²C specification, you must terminate all I²C transactions with a stop condition.

ni845xI2cScriptOpen

Purpose

Opens an empty I²C script to begin adding commands to.

Format

```
int32 ni845xI2cScriptOpen (uInt32 * pScriptHandle);
```

Inputs

None.

Outputs

`uInt32 * pScriptHandle`

A pointer to an unsigned 32-bit integer to store the new script handle in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptOpen` to create a new script to use with the NI-845*x* I²C Advanced API. Pass the reference to I²C script functions to create the script. Then, call `ni845xI2cScriptRun` to execute your script on your NI 845*x* device. After you finish executing your script, use `ni845xI2cScriptClose` to delete the script.

ni845xI2cScriptRead

Purpose

Adds an I²C Script Read command to an I²C script referenced by `ScriptHandle`. This command reads an array of data from an I²C slave device.

Format

```
int32 ni845xI2cScriptRead (
      uInt32   ScriptHandle,
      uInt32   NumBytesToRead,
      int32    NotAcknowledgeLastByte,
      uInt32*  pScriptReadIndex
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt32 NumBytesToRead`

The number of bytes to read. This must be nonzero.

`int32 NotAcknowledgeLastByte`

Whether the last byte read is acknowledged or not acknowledged by the I²C interface. If `NotAcknowledgeLastByte` is `kNi845xI2cNakTrue`, all bytes up to the last byte read are acknowledged. The last byte read is not acknowledged. If `NotAcknowledgeLastByte` is `kNi845xI2cNakFalse` (0), all bytes are acknowledged.

Outputs

`uInt32 * pScriptReadIndex`

An unsigned 32-bit integer pointer that stores the script read index. `pScriptReadIndex` is the read command index within the script. It is used as an input into `ni845xI2cScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use ni845xI2cScriptRead to add an I²C Script Read command to an I²C script referenced by ScriptHandle. This command reads an array of data from a device connected to the I²C port you specify when you use ni845xI2cScriptRun to execute the script. This command assumes that a start condition and address+read condition have been issued to the I²C bus using prior I²C script commands. It clocks in NumBytesToRead bytes from the I²C slave device, acknowledging each byte up to the last one. Depending on the type of I²C transaction you want to build, you may want to acknowledge (ACK) or not acknowledge (NAK) the last data byte read, which you can specify with the NotAcknowledgeLastByte input. To obtain the data read from the specified I²C port, you can pass the value of pScriptReadIndex after script execution to ni845xI2cScriptExtractReadDataSize to get the read data size and then to ni845xI2cScriptExtractReadData after script execution, which returns the data read by this script command.

ni845xI2cScriptReset

Purpose

Resets an I2C script referenced by `ScriptHandle` to an empty state.

Format

```
int32 ni845xI2cScriptReset (uInt32 ScriptHandle);
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptReset` to reset a script to an empty state. Any commands or read data stored in the script are deleted.

ni845xI2cScriptRun

Purpose

Sends the I^{2}C script to the desired NI 845*x* device, which then interprets and runs it.

Format

```
int32 ni845xI2cScriptRun  (
      uInt32 ScriptHandle,
      uInt32 DeviceHandle,
      uInt8  PortNumber
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

An unsigned byte that represents the port number to run the script on.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptRun` to execute an I^{2}C script indicated by `ScriptHandle` on the device indicated by `DeviceHandle`. You must first create an I^{2}C script using the I^{2}C scripting commands. Next, pass the script handle into `ScriptHandle`. If you have multiple NI 845*x* devices installed in your system, you can select which device to write your I^{2}C script to by passing its handle into `DeviceHandle`. If your NI 845*x* device supports multiple I^{2}C ports, you can also select which port to write your I^{2}C script to. For single I^{2}C port NI 845*x* devices, you must use the default port (0). In this way, you can create one script to run on various NI 845*x* devices, on various I^{2}C ports within those devices. `ni845xI2cScriptRun` loads and executes your I^{2}C script on the NI 845*x* device and I^{2}C port you specify, then returns success or error. If your script contained any read commands, you may use `ni845xI2cScriptExtractReadDataSize` to get the read data size, and `ni845xI2cScriptExtractReadData` to extract the read data after executing `ni845xI2cScriptRun`.

ni845xI2cScriptWrite

Purpose

Adds an I²C Script Write command to an I²C script referenced by `ScriptHandle`. This command writes an array of data to an I²C slave device.

Format

```
int32 ni845xI2cScriptWrite (
       uInt32  ScriptHandle,
       uInt32  WriteSize,
       uInt8 * pWriteData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xI2cScriptOpen`.

`uInt32 WriteSize`

The number of bytes to write. This must be nonzero.

`uInt8 * pWriteData`

A pointer to an array of bytes where the data to be written resides.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xI2cScriptWrite` to add an I²C Script Write command to an I²C script referenced by `ScriptHandle`. This command writes an array of data to an I²C slave device connected to the I²C port you specify when you use `ni845xI2cScriptRun` to execute the script. This command assumes that a start condition and address+write condition have been issued to the I²C bus using prior I²C script commands. It clocks the `pWriteData` array into the I²C slave device, testing for a slave device acknowledge after transmission of each byte. If a slave does not acknowledge a byte, `ni845xI2cScriptRun` exits with an error.

8

Using the NI-845*x* SPI API

This chapter helps you get started with the SPI API.

NI-845x SPI Basic Programming Model

The SPI Basic API provides the most fundamental SPI transaction type: write/read. You can access most off-the-shelf SPI devices using this transaction. The SPI Basic API allows you to easily and quickly develop applications to communicate with these devices. For those situations in which the SPI Basic API does not provide the functionality you need, use the SPI Advanced API to create custom SPI transactions.

When you use the SPI Basic API, the first step is to create an SPI configuration to describe the communication requirements between the 845*x* and the SPI device. To make an SPI configuration, create an SPI configuration reference and set the appropriate properties as desired. You can then read or write data to the SPI device.

The diagram in Figure 8-1 describes the programming model for the NI-845*x* SPI Basic API. Within the application, you repeat this programming model for each SPI device. The diagram is followed by a description of each step in the model.

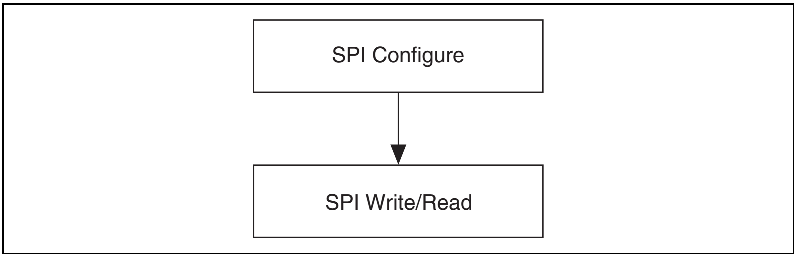


Figure 8-1. NI-845x SPI API Basic Programming Model

SPI Configure

Use the **NI-845x SPI Configuration Property Node** in LabVIEW and `ni845xSpiConfiguration*` calls in other languages to set the specific SPI configuration that describes the characteristics of the device to communicate with.

SPI Write Read

Use **NI-845x SPI Write Read.vi** in LabVIEW and `ni845xSpiWriteRead` in other languages to exchange an array of data with an SPI slave device.

SPI Timing Characteristics

Figure 8-2 and Tables 8-1 and 8-2 show the timing characteristics of the SPI bus when using the SPI Basic API. If the timing characteristics of your device do not fit within these parameters, you can use the SPI Advanced API to adjust the bus characteristics to match those of your device.

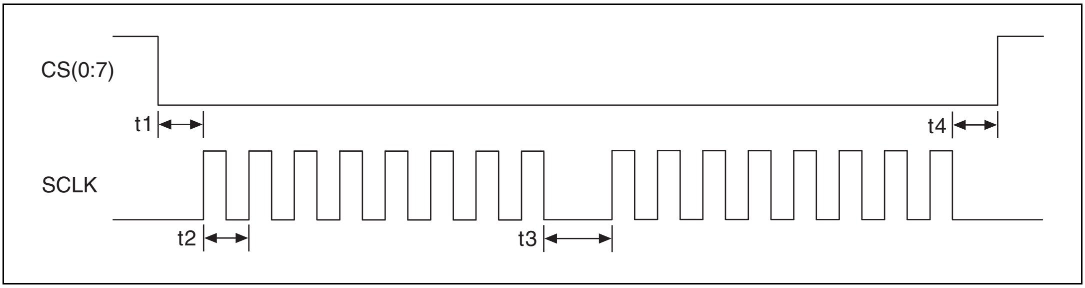


Figure 8-2. SPI Waveform

Table 8-1. NI USB-8451 Basic API SPI Timing Characteristics

Symbol	Parameter	Min	Max	Units
t1	CS(0:7) assertion to first SCLK edge	5	15.4	µs
t2	SCLK period	0.08333	20.83	µs
t3	SCLK setup time	8.5	19	µs
t4	Last SCLK edge to CS(0:7) deassertion	7.4	8.24	µs

Table 8-2. NI USB-8452 Basic API SPI Timing Characteristics

Symbol	Parameter	Min	Max	Units
t1	CS(0:7) assertion to first SCLK edge	2.2	1 + ½ t2[1]	µs
t2	SCLK period	0.02	1000	µs
t3	SCLK setup time	2.0	1 + ½ t2[1]	µs
t4	Last SCLK edge to CS(0:7) deassertion	2.2	1 + ½ t2[1]	µs
[1] If 1 + ½ t2 is less than the minimum specification, the maximum is 0.5 µs larger than the minimum.				

NI-845x SPI Advanced Programming Model

The SPI Advanced API provides a set of script commands that allow you great flexibility to construct custom SPI transactions to address your particular needs. For example, you can use scripting in the following scenarios:

- Executing individual byte transfers on the bus, with or without variable delays in between, so that you can observe device response.
- Issuing a transaction to a device and measuring its responses (using NI 845*x* DIO pins configured for input) at multiple points within the transaction.
- Doing performance testing, in which you see how a device responds to a variable delay, clock rate change, etc. between each byte transfer within a transaction.
- Gang programming a set of EEPROMs, then verifying the data by reading from each one afterwards.
- Communicating with devices that have an active high chip select line.

When you use the SPI Advanced API, the first step is to create a script that describes the communication between an SPI master and an SPI slave device. Then you execute the script and extract the read data if needed. The script size is limited only by the amount of memory available on your PC. The number of read commands, SPI Script Write Read, SPI Script DIO Read Port, and SPI Script DIO Read Line within each script is limited to 64.

The diagram in Figure 8-3 describes an example of programming with the scripting functions for the NI-845*x* SPI Advanced API. The diagram is followed by a description of each step in the model.

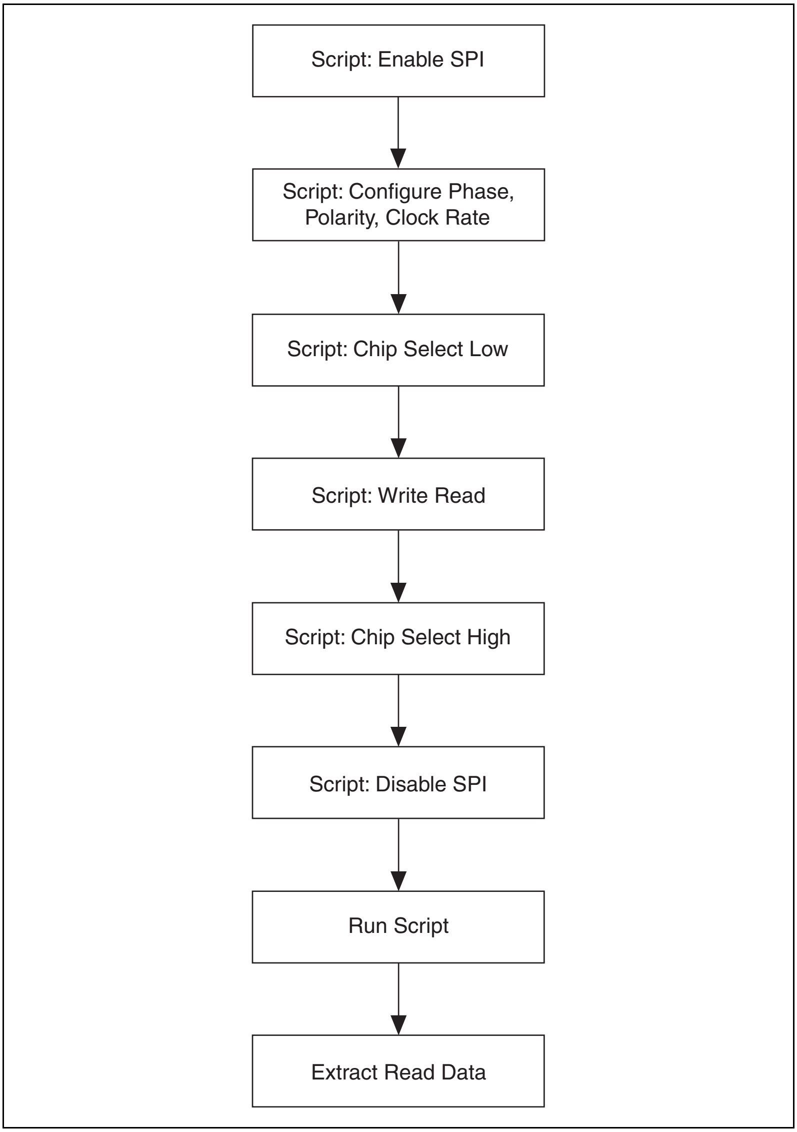


Figure 8-3. Scripting Functions Programming Example

Script: Enable SPI

Use **NI-845x SPI Script Enable SPI.vi** in LabVIEW and `ni845xSpiScriptEnableSPI` in other languages to add an SPI Script Enable SPI command to the SPI script. This command switches the pins on the SPI port you specify when you run the script from tristate to master mode function.

Script: Configure Phase, Polarity, Clock Rate

Use **NI-845x SPI Script Clock Polarity Phase.vi** in LabVIEW and `ni845xSpiScriptClockPolarityPhase` in other languages to add an SPI Script Clock Polarity Phase command to the SPI script. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA) for the SPI port you specify when you run the script.

Use **NI-845x SPI Script Clock Rate.vi** in LabVIEW and `ni845xSpiScriptClockRate` in other languages to add an SPI Script Clock Rate command to the SPI script. This command sets the SPI clock rate for the SPI port you specify when you run the script.

Script: Chip Select Low

Use **NI-845x SPI Script CS Low.vi** in LabVIEW and `ni845xSpiScriptCSLow` in other languages to add an SPI Script CS Low command to the SPI script. This command sets an SPI chip select to the logic low state when you run the script.

Script: Write Read

Use **NI-845x SPI Script Write Read.vi** in LabVIEW and `ni845xSpiScriptWriteRead` in other languages to add an SPI Script Write Read command to the SPI script. This command exchanges an array of data with an SPI slave device connected to the SPI port you specify when you run the script.

Script: Chip Select High

Use **NI-845x SPI Script CS High.vi** in LabVIEW and `ni845xSpiScriptCSHigh` in other languages to add an SPI Script CS High command to the SPI script. This command sets an SPI chip select to the logic high state when you run the script.

Script: Disable SPI

Use **NI-845x SPI Script Disable SPI.vi** in LabVIEW and `ni845xSpiScriptDisableSPI` in other languages to add an SPI Script Disable SPI command to the SPI script. This command tristates the pins on the SPI port you specify when you run the script.

Run Script

Use **NI-845x SPI Run Script.vi** in LabVIEW and `ni845xSpiScriptRun` in other languages to execute an SPI script on the desired device.

Extract Read Data

Use **NI-845x SPI Extract Script Read Data.vi** in LabVIEW and `ni845xSpiScriptExtractReadData` in other languages to extract the desired read data from a previously run SPI script. Each SPI script read command (SPI Script Read, SPI Script DIO Read Port, SPI Script DIO Read Line) returns a script read index to be passed into the Extract Read Data function.

9

NI-845*x* SPI API for LabVIEW

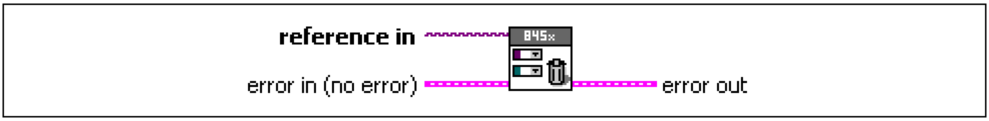
This chapter lists the LabVIEW VIs for the NI-845*x* SPI API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

General Device

NI-845x Close Reference.vi

Purpose

Closes a previously opened reference.



Inputs

reference in is a reference to an NI 845*x* device, I^{2}C configuration, SPI configuration, SPI stream configuration, I^{2}C script, or SPI script.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x Close Reference.vi** to close a previously opened reference.

NI-845x Device Property Node

Purpose

A property node with the NI-845*x* Device class preselected. This property node allows you to modify properties of your NI 845*x* device.

Inputs

device reference in is a reference to an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to an NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x Device Property Node**.

DIO:Active Port

The **DIO:Active Port** property sets the active DIO port for further DIO port configuration. The format for this property is a decimal string. For example, the string `0` represents DIO Port 0. The default value of this property is `0`. For NI 845*x* devices with one DIO port, the port value must be `0`.

DIO:Driver Type

The **DIO:Driver Type** property configures the active DIO port with the desired driver type characteristics. **DIO:Driver Type** uses the following values:

`Open-Drain`

The DIO driver type is configured for open-drain.

`Push-Pull`

The DIO driver type is configured for push-pull. The actual voltage driven (when sourcing a high value) is determined by the *I/O Voltage Level* property.

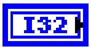
The default value of this property is `Push-Pull`.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available driver types on your hardware.

DIO:Line Direction Map

The **DIO:Line Direction Map** property sets the line direction map for the active DIO Port. The value is a bitmap that specifies the function of each individual line within the port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

The default value of this property is `0` (all lines configured for input).



I/O Voltage Level

The **I/O Voltage Level** property sets the board voltage. This property sets the voltage for SPI, I^2C, and DIO. The default value for this property is `3.3V`. This property uses the following values:

`3.3V`

I/O Voltage is set to 3.3 V.

`2.5V`

I/O Voltage is set to 2.5 V.

`1.8V`

I/O Voltage is set to 1.8 V.

`1.5V`

I/O Voltage is set to 1.5 V.

`1.2V`

I/O Voltage is set to 1.2 V.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

I^2C Pullup Enable

The **I^2C Pullup Enable** property enables or disables the internal pullup resistors connected to SDA and SCL.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your hardware has onboard pull-up resistors.

NI-845x Device Reference

Purpose

Specifies the device resource to be used for communication.

Description

Use the **NI-845x Device Reference** to describe the NI 845*x* device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845*x* API call to invoke the function on the associated NI 845*x* device.

Configuration

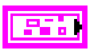
NI-845x SPI Configuration Property Node

Purpose

A property node with the NI-845*x* SPI Configuration class preselected. This property node allows you to query and modify SPI configuration properties of your NI 845*x* device.

Inputs

spi configuration in is a reference to a specific SPI configuration that describes the characteristics of the device to communicate with.



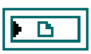
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

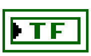
source identifies the VI where the error occurred.

Outputs



spi configuration out is a reference to a specific SPI configuration that describes the characteristics of the device to communicate with.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x SPI Configuration Property Node**.

Chip Select

Selects the chip select line for this configuration.

The default value for this property is `0`.

Port

Specifies the SPI port that this configuration communicates across.

The default value for this property is `0`.

Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the number of SPI ports your NI 845*x* device supports.

Clock Rate in kHz

Specifies the SPI clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports. If your hardware does not support the supplied clock rate, a warning is generated, and the next smallest supported clock rate is used. If the supplied clock rate is smaller than the smallest supported clock rate, an error is generated.

The default value for this property is `1000` kHz (`1` MHz).

Clock Polarity

Sets the idle state of the clock line for the SPI Port. **Clock Polarity** uses the following values:

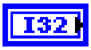
`0` (Idle Low)

Clock is low in the idle state.

`1` (Idle High)

Clock is high in the idle state.

The default value for this property is `0` (Idle Low).



Clock Phase

Sets the positioning of the data bits relative to the clock edges for the SPI Port. **Clock Phase** uses the following values:

`0` (First Edge)

Data is centered on the first edge of the clock period.

`1` (Second Edge)

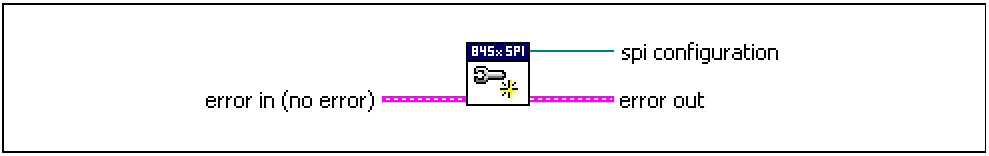
Data is centered on the second edge of the clock period.

The default value for this property is `0` (First Edge).

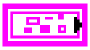
NI-845x SPI Create Configuration Reference.vi

Purpose

Creates a new NI-845*x* SPI configuration.



Inputs



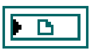
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

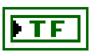
source identifies the VI where the error occurred.

Outputs

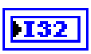


spi configuration is a reference to the newly created NI-845*x* SPI configuration.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

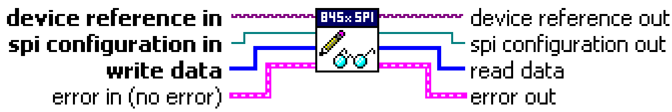
Use **NI-845x SPI Create Configuration Reference.vi** to create a new configuration to use with the NI-845*x* SPI Basic API. Pass the reference to a property node to make the configuration match the settings of your SPI slave. Then, pass the configuration to the SPI basic functions to execute them on the described SPI slave. After you finish communicating with your SPI slave, pass the reference into a new property node to reconfigure it or use **NI-845x Close Reference.vi** to delete the configuration.

Basic

NI-845x SPI Write Read.vi

Purpose

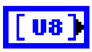
Exchanges an array of data with an SPI slave device.



Inputs


device reference in is a reference to an NI 845*x* device.


spi configuration in is a reference to a specific SPI configuration that describes the characteristics of the device to communicate with. Connect this configuration reference into a property node to set the specific configuration parameters.


write data contains an array of data to write to the SPI slave.


error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.


status is TRUE if an error occurred. This VI is not executed when status is TRUE.


code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.


source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

spi configuration out is a reference to the SPI configuration after this VI runs.

read data contains an array of read data from an SPI interface.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

> **status** is TRUE if an error occurred.
>
> **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.
>
> **source** identifies the VI where the error occurred.

Description

Use **NI-845x SPI Write Read.vi** to exchange an array of data with an SPI slave device. Due to the full-duplex nature of SPI, the size of the read data equals the size of the write data, unless there is an error. Some SPI devices act as receivers only and require one or more command and data bytes to be sent to them in one SPI transaction. As this is device specific, you need to review the device datasheet to package the required commands and data into the write data array. Other SPI devices act as transceivers. These devices can receive data much like receiver-only devices. But they can also transmit data, which usually requires writing one or more command bytes plus a number of bytes equal to the number of bytes desired to be read from the device. In most cases, the values of these bytes are not important, as they serve only to clock data out of the device. Here again, the SPI transaction formats are device specific, so you need to review the device datasheet to package the required commands and data into the write data array.

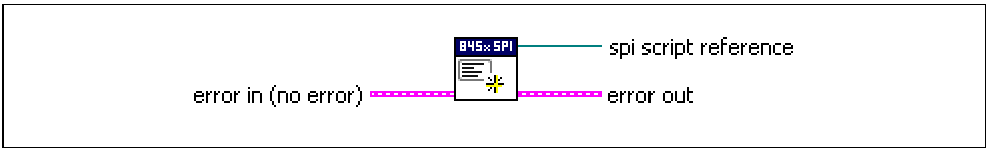
Before using **NI-845x SPI Write Read.vi**, you need to ensure that the configuration parameters specified in **spi configuration in** are correct for the device you currently want to access.

Advanced

NI-845x SPI Create Script Reference.vi

Purpose

Creates a new NI-845*x* SPI script.



Inputs

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference is a reference to the newly created NI-845*x* SPI script.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

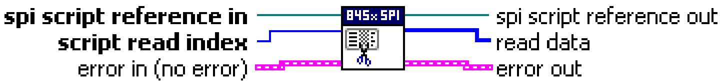
Description

Use **NI-845x SPI Create Script Reference.vi** to create a new script to use with the NI-845*x* SPI Advanced API. Pass the reference to SPI script functions to create the script. Then, call **NI-845x SPI Run Script.vi** to execute your script on your NI 845*x* device. After you have finished executing your script, use **NI-845x Close Reference.vi** to delete the script.

NI-845x SPI Extract Script Read Data.vi

Purpose

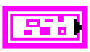
Extracts the desired read data from an SPI script, referenced by **spi script reference in**, which has been processed by **NI-845x SPI Run Script.vi**. Each script read command (**NI-845x SPI Script Write Read.vi**, **NI-845x SPI Script DIO Read Port.vi, NI-845x SPI Script DIO Read Line.vi**) returns a script read index. Data may be extracted for each script read index in a script, by wiring each to a separate **NI-845x SPI Extract Script Read Data.vi**.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

script read index identifies the read in the script whose data should be extracted.



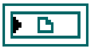
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

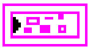
source identifies the VI where the error occurred.

Outputs



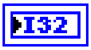
spi script reference out is a reference to an SPI script after this VI runs.

read data is the data returned for the script command specified by **script read index**.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

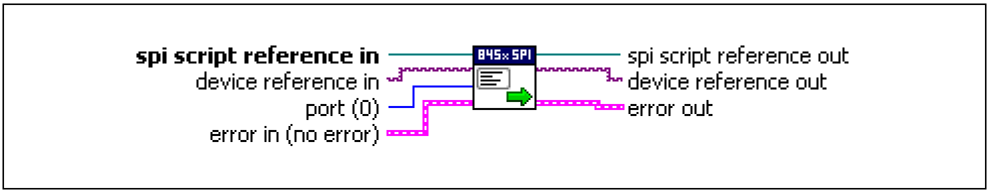
Use **NI-845x SPI Extract Script Read Data.vi** to extract the desired read data from an SPI script, referenced by **spi script reference in**, which has been processed by **NI-845x SPI Run Script.vi**. Each SPI script read command (**NI-845x SPI Script Write Read.vi**, **NI-845x SPI Script DIO Read Port.vi, NI-845x SPI Script DIO Read Line.vi**) returns a script read index.

Data may be extracted for each script read in different ways. For example, you can wire the script read index output of each script read VI to its own **NI-845x SPI Extract Script Read Data.vi**. You can also place **NI-845x SPI Extract Script Read Data.vi** in a For Loop and wire the loop iteration terminal to the **script read index** input. Add one to the script read index output of the last read and wire this value to the loop count terminal. The output of the For Loop will be an array of read data arrays.

NI-845x SPI Run Script.vi

Purpose

Executes an SPI script referenced by **spi script reference in** on the device referenced by **device reference in**.

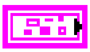


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

device reference in is a reference to an NI 845*x* device.

port specifies the SPI port this script will run on.



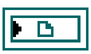
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



spi script reference out is a reference to the SPI script after this VI runs.

device reference out is a reference to the NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x SPI Run Script.vi** to execute an SPI script referenced by **spi script reference in** on the device referenced by **device reference in**. You must first create an SPI script using the SPI scripting VIs. Next, wire its script reference into **spi script reference in**.

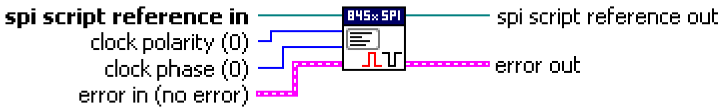
If you have multiple NI 845*x* devices installed in your system, you can select which device to write your SPI script to by wiring its device reference to **device reference in**. If your NI 845*x* device supports multiple SPI ports, you can also select which port to write your SPI script to. For single SPI port NI 845*x* devices, you must use the default port (`0`). In this way, you can create one script to run on various NI 845*x* devices, on various SPI ports within those devices.

NI-845x SPI Run Script.vi loads and executes your SPI script on the NI 845*x* device and SPI port you specify, then returns success or error. If your script contained any read commands, you may use **NI-845x SPI Extract Script Read Data.vi** to extract the read data after executing **NI-845x SPI Run Script.vi**.

NI-845x SPI Script Clock Polarity Phase.vi

Purpose

Adds an SPI Script Clock Polarity Phase command to an SPI script referenced by **spi script reference in**. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA).



Inputs

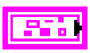
spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

clock polarity sets the idle state of the clock line. The values for **clock polarity** are:

0 (Idle Low)	low in idle state
1 (Idle High)	high in idle state

clock phase sets the positioning of the data bits relative to the clock edges. The values for **clock phase** are:

0 (First Edge)	data centered on first edge of clock period
1 (Second Edge)	data centered on second edge of clock period



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

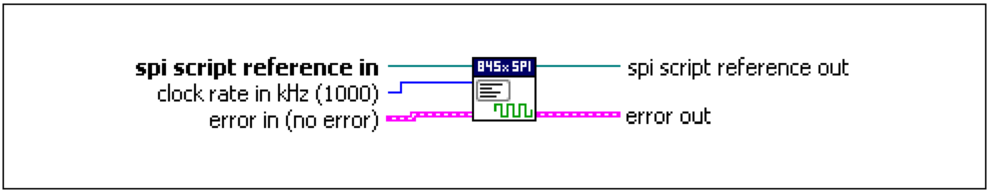
Use **NI-845x SPI Script Clock Polarity Phase.vi** to add an SPI Script Clock Polarity Phase command to an SPI script referenced by **spi script reference in**. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA) for the SPI port you specify when you use **NI-845x SPI Run Script.vi** to execute the script.

Clock polarity sets the idle state of the SPI clock line. The default (`0`) sets the clock line to idle at a low logic level. Setting the clock polarity to `1` sets the clock line to idle at a high logic level. **Clock phase** sets the SPI clock edge on which the NI 845*x* SPI port centers each MOSI data bit. The default (`0`) centers each MOSI data bit on the first edge of each clock cycle. Setting the clock phase to `1` causes each MOSI data bit to be centered on the second edge of each clock cycle.

NI-845x SPI Script Clock Rate.vi

Purpose

Adds an SPI Script Clock Rate command to an SPI script referenced by **spi script reference in**. This command sets the SPI clock rate.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

clock rate in kHz specifies the SPI clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, which clock rates your NI 845*x* device supports.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

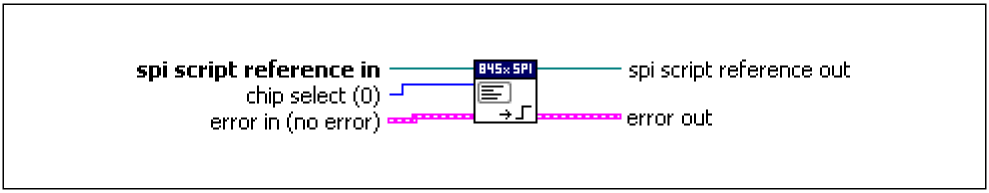
Description

Use **NI-845x SPI Script Clock Rate.vi** to add an SPI Script Clock Rate command to an SPI script referenced by **spi script reference in**. This command sets the SPI clock rate for the SPI port you specify when you use **NI-845x SPI Run Script.vi** to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845*x* software adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

NI-845x SPI Script CS High.vi

Purpose

Adds an SPI Script CS High command to an SPI script referenced by **spi script reference in**. This command sets an SPI chip select to the logic high state.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

chip select specifies the chip select to set high.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

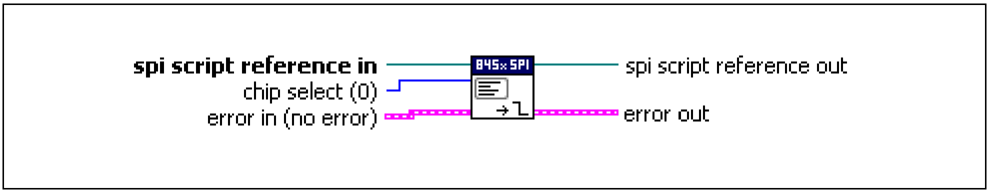
Description

Use **NI-845x SPI Script CS High.vi** to add an SPI Script CS High command to an SPI script referenced by **spi script reference in**. This command sets an SPI chip select to the logic high state.

NI-845x SPI Script CS Low.vi

Purpose

Adds an SPI Script CS Low command to an SPI script referenced by **spi script reference in**. This command sets an SPI chip select to the logic low state.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

chip select specifies the chip select to set low.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

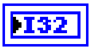
source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

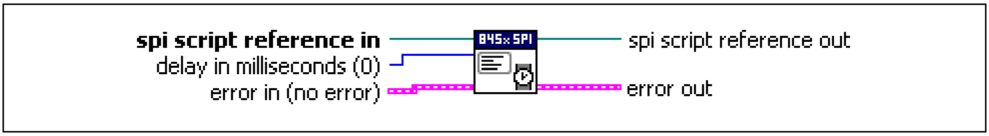
Description

Use **NI-845x SPI Script CS Low.vi** to add an SPI Script CS Low command to an SPI script referenced by **spi script reference in**. This command sets an SPI chip select to the logic low state.

NI-845x SPI Script Delay.vi

Purpose

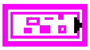
Adds an SPI Script Delay command to an SPI script referenced by **spi script reference in**. This command adds a delay after the previous SPI script command.



Inputs

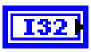
spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

delay in milliseconds specifies the desired delay.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

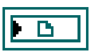
status is TRUE if an error occurred. This VI is not executed when status is TRUE.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

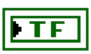
source identifies the VI where the error occurred.

Outputs



spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute

the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

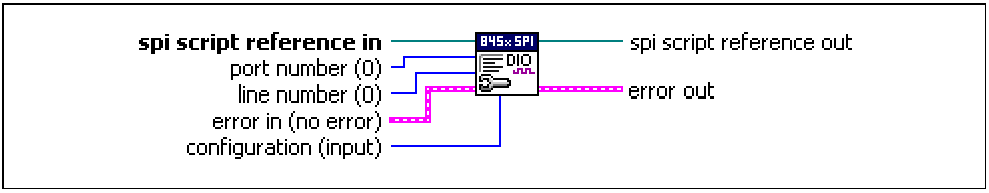
Description

Use **NI-845x SPI Script Delay.vi** to add an SPI Script Delay command to an SPI script referenced by **spi script reference in**. This command adds a delay after the previous SPI script command.

NI-845x SPI Script DIO Configure Line.vi

Purpose

Adds an SPI Script DIO Configure Line command to an SPI script referenced by **spi script reference in**. This command configures a DIO line on an NI 845*x* device.

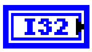


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

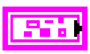
port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to configure.



configuration specifies the line configuration. **configuration** uses the following values:

- **input** The line is configured for input.
- **output** The line is configured for output.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

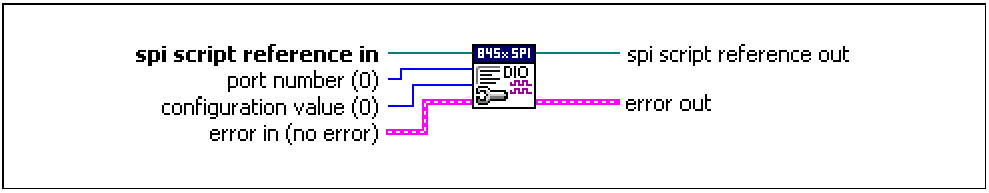
Description

Use **NI-845x SPI Script DIO Configure Line.vi** to add an SPI Script DIO Configure Line command to an SPI script referenced by **spi script reference in**. This command allows you to configure one line, specified by **line number**, of a byte-wide DIO port, as in input or output. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x SPI Script DIO Configure Port.vi

Purpose

Adds an SPI Script DIO Configure Port command to an SPI script referenced by **spi script reference in**. This command configures a DIO port on an NI 845*x* device.

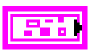


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

port number specifies the DIO port to configure.

configuration value is a bitmap that specifies the function of each individual line of a port. If bit *x* = 1, line *x* is an output. If bit *x* = 0, line *x* is an input.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

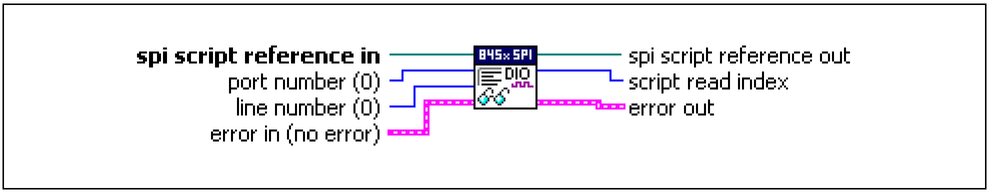
Description

Use **NI-845x SPI Script DIO Configure Port.vi** to add an SPI Script DIO Configure Port command to an SPI script referenced by **spi script reference in**. This command allows you to configure all eight lines of a byte-wide DIO port. Setting a bit to `1` configures the corresponding DIO port line for output. Setting a bit to `0` configures the corresponding port line for input. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x SPI Script DIO Read Line.vi

Purpose

Adds an SPI Script DIO Read Line command to an SPI script referenced by **spi script reference in**. This command reads from a DIO port on an NI 845*x* device.

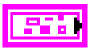


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

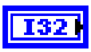
port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

script read index is the index of the read command within the script. It is used as an input into **NI-845x SPI Extract Script Read Data.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

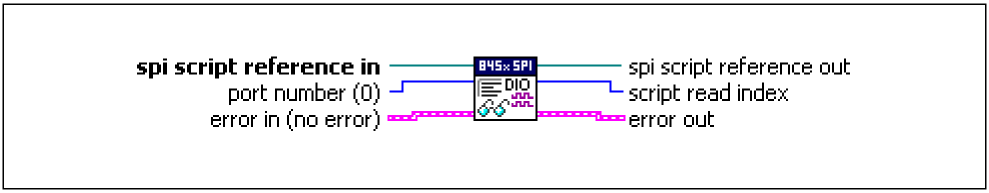
Use **NI-845x SPI Script DIO Read Line.vi** to add an SPI Script DIO Read command to an SPI script referenced by **spi script reference in**. This command allows you to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

To obtain the logic level read from the specified DIO port line, wire **script read index** to **NI-845x SPI Extract Script Read Data.vi** after script execution. If **NI-845x SPI Extract Script Read Data.vi** returns `0`, the logic level read on the specified line was low. If **NI-845x SPI Extract Script Read Data.vi** returns `1`, the logic level read on the specified line was high.

NI-845x SPI Script DIO Read Port.vi

Purpose

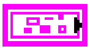
Adds an SPI Script DIO Read Port command to an SPI script referenced by **spi script reference in**. This command reads from a DIO port on an NI 845*x* device.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

port number specifies the DIO port to read.



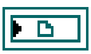
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

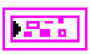
source identifies the VI where the error occurred.

Outputs



spi script reference out is a reference to the SPI script after this VI runs.

script read index is the index of the read command within the script. It is used as an input into **NI-845x SPI Extract Script Read Data.vi**.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

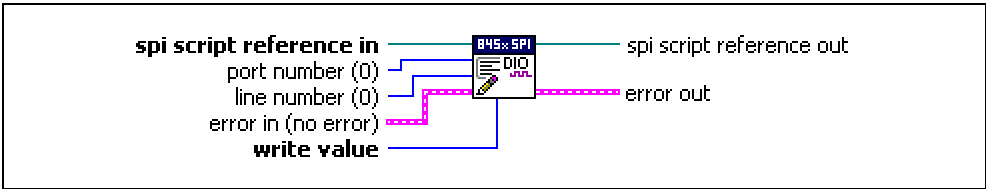
Use **NI-845x SPI Script DIO Read Port.vi** to add an SPI Script DIO Read Port command to an SPI script referenced by **spi script reference in**. This command allows you to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

To obtain the data byte read from the specified DIO port, wire **script read index** to **NI-845x SPI Extract Script Read Data.vi** after script execution, which returns the data byte read by this script command.

NI-845x SPI Script DIO Write Line.vi

Purpose

Adds an SPI Script DIO Write Line command to an SPI script referenced by **spi script reference in**. This command writes to a DIO line on an NI 845*x* device.

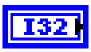


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

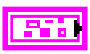
port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to write.



write value specifies the value to write to the line. **write value** uses the following values:

`0` (Logic Low)	The line is set to the logic low state.
`1` (Logic High)	The line is set to the logic high state.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

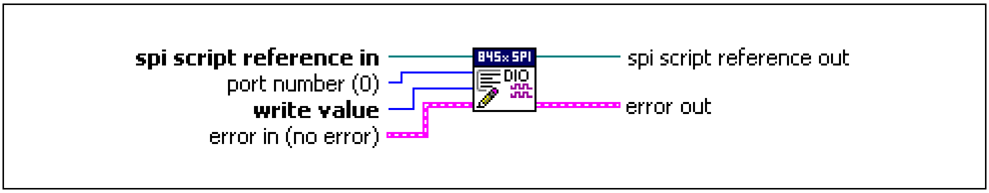
Description

Use **NI-845x SPI Script DIO Write Line.vi** to add an SPI Script DIO Write command to an SPI script referenced by **spi script reference in**. This command allows you to write one line, specified by **line number**, of a byte-wide DIO port. If **write value** is `1`, the specified line's output is driven to a high logic level. If **write value** is `0`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x SPI Script DIO Write Port.vi

Purpose

Adds an SPI Script DIO Write Port command to an SPI script referenced by **spi script reference in**. This command writes to a DIO port on an NI 845*x* device.

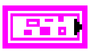


Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

port number specifies the DIO port to write.

write value is the value to write to the DIO port. Only lines configured for output are updated.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

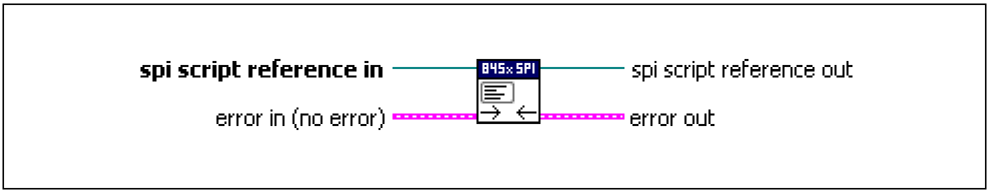
Description

Use **NI-845x SPI Script DIO Write Port.vi** to add an SPI Script DIO Write Port command to an SPI script referenced by **spi script reference in**. This command allows you to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x SPI Script Disable SPI.vi

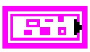
Purpose

Adds an SPI Script Disable SPI command to an SPI script referenced by **spi script reference in**. This command tristates the pins on an SPI port specified using **NI-845x SPI Run Script.vi**. It also tristates all chip select pins.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.



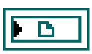
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

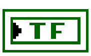
source identifies the VI where the error occurred.

Outputs

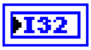


spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

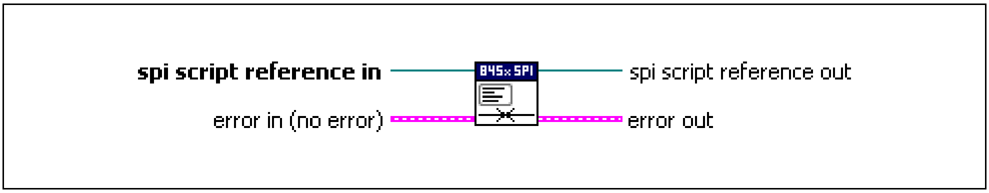
Description

Use **NI-845x SPI Script Disable SPI.vi** to add an SPI Script Disable SPI command to an SPI script referenced by **spi script reference in**. This command tristates the pins on the SPI port you specify when you use **NI-845x SPI Run Script.vi**. All chip select pins are also tristated.

NI-845x SPI Script Enable SPI.vi

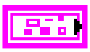
Purpose

Adds an SPI Script Enable SPI command to an SPI script referenced by **spi script reference in**. This command switches the pins on an SPI port specified using **NI-845x SPI Run Script.vi** to master mode function. All chip select pins are switched from tristate to push-pull output driven high.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.



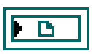
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

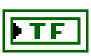
source identifies the VI where the error occurred.

Outputs

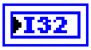


spi script reference out is a reference to the SPI script after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

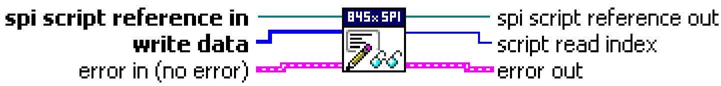
Use **NI-845*x* SPI Script Enable SPI.vi** to add an SPI Script Enable SPI command to an SPI script referenced by **spi script reference in**. This command switches the pins on the SPI port you specify when you use **NI-845x SPI Run Script.vi**, from tristate to master mode function.

Also, all chip select pins are switched from tristate to push-pull output driven high. It is important to keep this in mind if you are creating a script to access a device with an active high chip select input. You need to enable SPI and write the device chip select low until you want to access it, at which time you set the chip select high, perform the write/read, and then set the chip select low.

NI-845x SPI Script Write Read.vi

Purpose

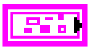
Adds an SPI Script Write Read command to an SPI script referenced by **spi script reference in**. This command exchanges an array of data with an SPI slave device.



Inputs

spi script reference in is a reference to an SPI script that is run on an NI 845*x* device.

write data contains an array of data to write to the SPI slave.



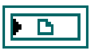
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

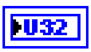
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

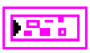
Outputs



spi script reference out is a reference to the SPI script after this VI runs.



script read index is the index of the write/read command within the script. It is used as an input into **NI-845x SPI Extract Script Read Data.vi**.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x SPI Script Write Read.vi** to add an SPI Script Write Read command to an SPI script referenced by **spi script reference in**. This command exchanges an array of data with an SPI slave device connected to the SPI port you specify when you use **NI-845x SPI Run Script.vi** to execute the script.

Due to the full-duplex nature of SPI, the size of the read data equals the size of the write data, unless there is an error. Some SPI devices act as receivers only and require one or more command and data bytes to be sent to them in one SPI transaction. As this is device specific, you need to review the device datasheet to package the required commands and data into the write data array. Other SPI devices act as transceivers. These devices can receive data much like receiver-only devices. But they can also transmit data, which usually requires writing one or more command bytes plus a number of bytes equal to the number of bytes desired to be read from the device. In most cases, the values of these bytes are not important, as they serve only to clock data out of the device. Here again, the SPI transaction formats are device specific, so you need to review the device datasheet to package the required commands and data into the write data array.

To obtain the data read from the specified SPI port, wire **script read index** to **NI-845x SPI Extract Script Read Data.vi** after script execution, which returns the data read by this script command.

10

NI-845*x* SPI API for C

This chapter lists the functions for the NI-845*x* SPI API for C and describes the format, purpose, and parameters for each function. The functions are listed alphabetically in four categories: general device, configuration, basic, and advanced.

Section Headings

The NI-845*x* SPI API for C functions include the following section headings.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Inputs and Outputs

These sections list the function input and output parameters.

Description

The description section gives details about the purpose and effect of each function.

Data Types

The NI-845*x* SPI API for C functions use the following data types.

Data Type	Purpose
`uInt8`	8-bit unsigned integer
`uInt16`	16-bit unsigned integer
`uInt32`	32-bit unsigned integer
`int8`	8-bit signed integer

Data Type	Purpose
`int16`	16-bit signed integer
`int32`	32-bit signed integer
`uInt8 *`	Pointer to an 8-bit unsigned integer
`uInt16 *`	Pointer to a 16-bit unsigned integer
`uInt32 *`	Pointer to a 32-bit unsigned integer
`int8 *`	Pointer to an 8-bit signed integer
`int16 *`	Pointer to a 16-bit signed integer
`int32 *`	Pointer to a 32-bit signed integer
`char *`	ASCII string represented as an array of characters terminated by null character (`'\0'`)

List of Functions

The following table contains an alphabetical list of the NI-845*x* SPI API for C functions.

Function	Purpose
`ni845xClose`	Closes a previously opened NI 845*x* device.
`ni845xCloseFindDeviceHandle`	Closes the handles created by `ni845xFindDevice`.
`ni845xDeviceLock`	Locks NI 845*x* devices for access by a single thread.
`ni845xDeviceUnlock`	Unlocks NI 845*x* devices.
`ni845xFindDevice`	Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.
`ni845xFindDeviceNext`	Finds subsequent devices after `ni845xFindDevice` has been called.
`ni845xOpen`	Opens an NI 845*x* device for use with various write, read, and device property functions.

Function	Purpose
ni845xSetIoVoltageLevel	Sets the voltage level of the NI-845*x* I/O pins (DIO/SPI/VioRef).
ni845xSpiConfigurationClose	Closes a previously opened configuration.
ni845xSpiConfigurationGetChipSelect	Retrieves the configuration chip select value.
ni845xSpiConfigurationGetClockPhase	Retrieves the configuration clock phase.
ni845xSpiConfigurationGetClockPolarity	Retrieves the configuration clock polarity.
ni845xSpiConfigurationGetClockRate	Retrieves the configuration clock rate in kilohertz.
ni845xSpiConfigurationGetPort	Retrieves the configuration port value.
ni845xSpiConfigurationOpen	Creates a new NI-845*x* SPI configuration.
ni845xSpiConfigurationSetChipSelect	Sets the configuration chip select.
ni845xSpiConfigurationSetClockPhase	Sets the configuration clock phase.
ni845xSpiConfigurationSetClockPolarity	Sets the configuration clock polarity.
ni845xSpiConfigurationSetClockRate	Sets the configuration clock rate in kilohertz.
ni845xSpiConfigurationSetPort	Sets the configuration port number.
ni845xSpiScriptClockPolarityPhase	Adds an SPI Script Clock Polarity Phase command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA).
ni845xSpiScriptClockRate	Adds an SPI Script Clock Rate command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock rate in kilohertz.
ni845xSpiScriptClose	Closes a previously opened script handle.
ni845xSpiScriptCSHigh	Adds an SPI Script CS High command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic high state.

Function	Purpose
`ni845xSpiScriptCSLow`	Adds an SPI Script CS Low command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic low state.
`ni845xSpiScriptDelay`	Adds an SPI Script Delay command to an SPI script referenced by `ScriptHandle`. This command adds a delay after the previous SPI script command.
`ni845xSpiScriptDioConfigureLine`	Adds an SPI Script DIO Configure Line command to an SPI script referenced by `ScriptHandle`. This command configures a DIO line on an NI 845*x* device.
`ni845xSpiScriptDioConfigurePort`	Adds an SPI Script DIO Configure Port command to an SPI script referenced by `ScriptHandle`. This command configures a DIO port on an NI 845*x* device.
`ni845xSpiScriptDioReadLine`	Adds an SPI Script DIO Read Line command to an SPI script referenced by `ScriptHandle`. This command reads from a DIO line on an NI 845*x* device.
`ni845xSpiScriptDioReadPort`	Adds an SPI Script DIO Read Port command to an SPI script referenced by `ScriptHandle`. This command reads from a DIO port on an NI 845*x* device.
`ni845xSpiScriptDioWriteLine`	Adds an SPI Script DIO Write Line command to an SPI script referenced by `ScriptHandle`. This command writes to a DIO line on an NI 845*x* device.
`ni845xSpiScriptDioWritePort`	Adds an SPI Script DIO Write Port command to an SPI script referenced by `ScriptHandle`. This command writes to a DIO port on an NI 845*x* device.

Function	Purpose
ni845xSpiScriptDisableSPI	Adds an SPI Script Disable SPI command to an SPI script referenced by ScriptHandle. This command tristates the pins on an SPI port specified using ni845xSpiScriptRun. It also tristates all chip select pins.
ni845xSpiScriptEnableSPI	Adds an SPI Script Enable SPI command to an SPI script referenced by ScriptHandle. This command switches the pins on an SPI port specified using ni845xSpiScriptRun to master mode function. All chip select pins are switched from tristate to push-pull output driven high.
ni845xSpiScriptExtractReadData	Extracts the desired read data from an SPI script, referenced by ScriptHandle, which has been processed by ni845xSpiScriptRun. Each script read command (ni845xSpiScriptWriteRead, ni845xSpiScriptDioReadPort, ni845xSpiScriptDioReadLine) returns a script read index. You can extract data for each script read index in a script, by passing each index to a separate call of ni845xSpiScriptExtractReadData.
ni845xSpiScriptExtractReadDataSize	Retrieves the read data size from an SPI script, referenced by ScriptHandle, which has been processed by ni845xSpiScriptRun. Each script read command (ni845xSpiScriptWriteRead, ni845xSpiScriptDioReadPort, ni845xSpiScriptDioReadLine) returns a script read index. You can extract data for each script read index in a script, by passing each index to ni845xSpiScriptExtractReadData.
ni845xSpiScriptOpen	Creates a new NI-845*x* SPI script.

Function	Purpose
ni845xSpiScriptReset	Resets an SPI script referenced by ScriptHandle to an empty state.
ni845xSpiScriptRun	Sends the SPI script to the desired NI 845*x* device, which then interprets and runs it.
ni845xSpiScriptWriteRead	Adds an SPI Script Write Read command to an SPI script referenced by ScriptHandle. This command exchanges an array of data with an SPI slave device.
ni845xSpiWriteRead	Exchanges an array of data with an SPI slave device.
ni845xStatusToString	Converts a status code into a descriptive string.

General Device

ni845xClose

Purpose

Closes a previously opened NI 845*x* device.

Format

```
int32 ni845xClose(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be closed.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xClose` to close a device handle previously opened by `ni845xOpen`. Passing an invalid handle to `ni845xClose` is ignored.

ni845xCloseFindDeviceHandle

Purpose

Closes the handles created by `ni845xFindDevice`.

Format

```
int32 ni845xCloseFindDeviceHandle (
      uInt32 FindDeviceHandle
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xCloseFindDeviceHandle` to close a find list. In this process, all allocated data structures are freed.

ni845xDeviceLock

Purpose

Locks NI 845*x* devices for access by a single thread.

Format

```
int32 ni845xDeviceLock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be locked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

This function locks NI 845*x* devices and prevents multiple processes or threads from accessing the device until the process or thread that owns the device lock calls an equal number of `ni845xDeviceUnlock` calls. Any thread or process that attempts to call `ni845xDeviceLock` when the device is already locked is forced to sleep by the operating system. This is useful for when multiple Basic API device accesses must occur uninterrupted by any other processes or threads. If a thread exits without fully unlocking the device, the device is unlocked. If a thread is the current owner of the lock, and calls `ni845xDeviceLock` again, the thread will not deadlock itself, but care must be taken to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` called. This function can possibly lock a device indefinitely: If a thread never calls `ni845xDeviceUnlock`, or fails to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` call, and never exits, other processes and threads are forced to wait. This is *not* recommended for users unfamiliar with threads or processes. A simpler alternative is to use scripts. Scripts provide the same capability to ensure transfers are uninterrupted, and with possible performance benefits.

ni845xDeviceUnlock

Purpose

Unlocks NI 845*x* devices.

Format

```
int32 ni845xDeviceUnlock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be unlocked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDeviceUnlock` to unlock access to an NI 845*x* device previously locked with `ni845xDeviceLock`. Every call to `ni845xDeviceLock` must have a corresponding call to `ni845xDeviceUnlock`. Refer to `ni845xDeviceLock` for more details regarding how to use device locks.

ni845xFindDevice

Purpose

Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.

Format

```
int32 ni845xFindDevice (
      char   * pFirstDevice,
      uInt32 * pFindDeviceHandle,
      uInt32 * pNumberFound
      );
```

Inputs

None.

Outputs

`char * pFirstDevice`

A pointer to the string containing the first NI 845*x* device found. You can pass this name to the `ni845xOpen` function to open the device. If no devices exist, this is an empty string.

`uInt32 * pFindDeviceHandle`

Returns a handle identifying this search session. This handle is used as an input in `ni845xFindDeviceNext` and `ni845xCloseFindDeviceHandle`.

`uInt32 * pNumberFound`

A pointer to the total number of NI 845*x* devices found in the system. You can use this number in conjunction with the `ni845xFindDeviceNext` function to find a particular device. If no devices exist, this returns 0.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDevice` to get a single NI 845*x* device and the number of NI 845*x* devices in the system. You can then pass the string returned to `ni845xOpen` to access the device. If you must discover more devices, use `ni845xFindDeviceNext` with `pFindDeviceHandle` and pNumberFound to find the remaining NI 845*x* devices in the system. After finding all desired devices, call ni845xCloseFindDeviceHandle to close the device handle and relinquish allocated resources.

Note pFirstDevice must be at least 256 bytes.

Note pFindDeviceHandle and pNumberFound are optional parameters. If only the first match is important, and the total number of matches is not needed, you can pass in a NULL pointer for both of these parameters, and the NI-845x driver automatically calls ni845xCloseFindDeviceHandle before this function returns.

ni845xFindDeviceNext

Purpose

Finds subsequent devices after `ni845xFindDevice` has been called.

Format

```
int32 ni845xFindDeviceNext (
      uInt32 FindDeviceHandle,
      char * pNextDevice
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

`char * pNextDevice`

A pointer to the string containing the next NI 845*x* device found. This is empty if no further devices are left.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDeviceNext` after first calling `ni845xFindDevice` to find the remaining devices in the system. You can then pass the string returned to `ni845xOpen` to access the device.

Note `pNextDevice` must be at least 256 bytes.

ni845xOpen

Purpose

Opens an NI 845*x* device for use with various write, read, and device property functions.

Format

```
int32 ni845xOpen (
      char   * pResourceName,
      uInt32 * pDeviceHandle
      );
```

Inputs

`char * pResourceName`

A resource name string corresponding to the NI 845*x* device to be opened.

Outputs

`uInt32 * pDeviceHandle`

A pointer to the device handle.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xOpen` to open an NI 845*x* device for access. The string passed to `ni845xOpen` can be any of the following: an `ni845xFindDevice` device string, an `ni845xFindDeviceNext` device string, a Measurement & Automation Explorer resource name, or a Measurement & Automation Explorer alias.

ni845xSetIoVoltageLevel

Purpose

Modifies the voltage output from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSetIoVoltageLevel (
       uInt32 DeviceHandle,
       uInt8  VoltageLevel
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 VoltageLevel`

The desired voltage level. `VoltageLevel` uses the following values:

- `kNi845x33Volts` (33): The output I/O high level is 3.3 V.
- `kNi845x25Volts` (25): The output I/O high level is 2.5 V.
- `kNi845x18Volts` (18): The output I/O high level is 1.8 V.
- `kNi845x15Volts` (15): The output I/O high level is 1.5 V.
- `kNi845x12Volts` (12): The output I/O high level is 1.2 V.

The default value of this property is 3.3 V.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSetIoVoltageLevel` to modify the board reference voltage of the NI 845*x* device. The board reference voltage is used for SPI, I^2C, and DIO. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

ni845xStatusToString

Purpose

Converts a status code into a descriptive string.

Format

```
void ni845xStatusToString (
       int32  StatusCode,
       uInt32 MaxSize,
       int8 * pStatusString
       );
```

Inputs

`int32 StatusCode`

Status code returned from an NI-845*x* function.

`uInt32 MaxSize`

Size of the `pStatusString` buffer (in bytes).

Outputs

`int8 * pStatusString`

ASCII string that describes `StatusCode`.

Description

When the status code returned from an NI-845*x* function is nonzero, an error or warning is indicated. This function obtains a description of the error/warning for debugging purposes.

The return code is passed into the `StatusCode` parameter. The `MaxSize` parameter indicates the number of bytes available in `pStatusString` for the description (including the NULL character). The description is truncated to size `MaxSize` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `String` is null-terminated, so you can use it with ANSI C functions such as `printf`.

For applications written in C or C++, each NI-845*x* function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845*x* use of this status.

NI-845*x* Status Codes

Status Code	Meaning
Negative	Error—Function did not perform expected behavior.
Positive	Warning—Function executed, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every NI-845*x* function. If an error is detected, you should close all NI-845*x* handles, then exit the application. If a warning is detected, you can display a message for debugging purposes, or simply ignore the warning.

In some situations, you may want to check for specific errors in the code and continue communication when they occur. For example, when communicating to an I^2C EEPROM, you may expect the device to NAK its address during a write cycle, and you may use this knowledge to poll for when the write cycle has completed.

Configuration

ni845xSpiConfigurationClose

Purpose

Closes a previously opened configuration.

Format

```
int32 ni845xSpiConfigurationClose (
      uInt32 ConfigurationHandle
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationClose` to close a previously opened configuration handle. Invalid configuration handles are ignored.

ni845xSpiConfigurationGetChipSelect

Purpose

Retrieves the configuration chip select value.

Format

```
int32 ni845xSpiConfigurationGetChipSelect (
        uInt32   ConfigurationHandle,
        uInt32 * pChipSelect
        );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

`uInt32 * pChipSelect`

A pointer to an unsigned 32-bit integer to store the chip select value in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationGetChipSelect` to retrieve the chip select stored in the configuration.

ni845xSpiConfigurationGetClockPhase

Purpose

Retrieves the configuration clock phase.

Format

```
int32 ni845xSpiConfigurationGetClockPhase (
       uInt32  ConfigurationHandle,
       int32 * pClockPhase
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

`int32 * pClockPhase`

A pointer to an integer to store the clock phase in. `pClockPhase` uses the following values:

- `kNi845xSpiClockPhaseFirstEdge` (0): Data is centered on the first edge of the clock period.
- `kNi845xSpiClockPhaseSecondEdge` (1): Data is centered on the second edge of the clock period.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationGetClockPhase` to retrieve the value of the clock phase that `ConfigurationHandle` uses.

ni845xSpiConfigurationGetClockPolarity

Purpose

Retrieves the configuration clock polarity.

Format

```
int32 ni845xSpiConfigurationGetClockPolarity (
       uInt32 ConfigurationHandle,
       int32 * pClockPolarity
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

`int32 * pClockPolarity`

A pointer to an integer to store the clock polarity in. `pClockPolarity` uses the following values:

- `kNi845xSpiClockPolarityIdleLow` (0): Clock is low in the idle state.
- `kNi845xSpiClockPolarityIdleHigh` (1): Clock is high in the idle state.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationGetClockPolarity` to retrieve the value of the clock polarity that the `ConfigurationHandle` uses to communicate with.

ni845xSpiConfigurationGetClockRate

Purpose

Retrieves the configuration clock rate in kilohertz.

Format

```
int32 ni845xSpiConfigurationGetClockRate (
       uInt32  ConfigurationHandle,
       uInt16 * pClockRate
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

`uInt16 * pClockRate`

A pointer to an unsigned 16-bit integer to store the clock rate in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationGetClockRate` to retrieve the SPI clock rate in kilohertz that the `ConfigurationHandle` runs at.

ni845xSpiConfigurationGetPort

Purpose

Retrieves the configuration port value.

Format

```
int32 ni845xSpiConfigurationGetPort (
      uInt32 ConfigurationHandle,
      uInt8 * pPort
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

Outputs

`uInt8 * pPort`

A pointer to an unsigned byte to store the port value in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationGetPort` to retrieve the SPI port that the `ConfigurationHandle` communicates across.

ni845xSpiConfigurationOpen

Purpose

Creates a new NI-845*x* SPI configuration.

Format

```
int32 ni845xSpiConfigurationOpen (
      uInt32 * pConfigurationHandle
      );
```

Inputs

None.

Outputs

`uInt32 * pConfigurationHandle`

A pointer to an unsigned 32-bit integer to store the configuration handle in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationOpen` to create a new configuration to use with the NI-845*x* SPI Basic API. Pass the configuration handle to the `ni845xSpiConfigurationSet*` series of functions to make the configuration match the settings of your SPI slave. Then, pass the configuration handle to the SPI basic functions to execute them on the described SPI slave. After you finish communicating with your SPI slave, pass the configuration handle to the `ni845xSpiConfigurationSet*` series of functions to reconfigure it or use `ni845xSpiConfigurationClose` to delete the configuration.

ni845xSpiConfigurationSetChipSelect

Purpose

Sets the configuration chip select.

Format

```
int32 ni845xSpiConfigurationSetChipSelect (
      uInt32 ConfigurationHandle,
      uInt32 ChipSelect
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`uInt32 ChipSelect`

Selects the chip select line for this configuration.

The default value for the chip select is 0.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationSetChipSelect` to select the chip select where the SPI slave device resides.

ni845xSpiConfigurationSetClockPhase

Purpose

Sets the configuration clock phase.

Format

```
int32 ni845xSpiSetConfigurationClockPhase (
       uInt32 ConfigurationHandle,
       int32  ClockPhase
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`int32 ClockPhase`

Sets the positioning of the data bits relative to the clock edges for the SPI Port. `ClockPhase` uses the following values:

- `kNi845xSpiClockPhaseFirstEdge` (0): Data is centered on the first edge of the clock period.
- `kNi845xSpiClockPhaseSecondEdge` (1): Data is centered on the second edge of the clock period.

The default value for this property is `kNi845xSpiClockPhaseFirstEdge`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationSetClockPhase` to set the clock phase to use when communicating with an SPI slave device.

ni845xSpiConfigurationSetClockPolarity

Purpose

Sets the configuration clock polarity.

Format

```
int32 ni845xSpiConfigurationSetClockPolarity (
      uInt32 ConfigurationHandle,
      int32  ClockPolarity
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`int32 ClockPolarity`

Sets the clock line idle state for the SPI Port. `ClockPolarity` uses the following values:

- `kNi845xSpiClockPolarityIdleLow` (0): Clock is low in the idle state.
- `kNi845xSpiClockPolarityIdleHigh` (1): Clock is high in the idle state.

The default value for this property is `kNi845xSpiClockPolarityIdleLow`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationSetClockPolarity` to set the clock polarity to use when communicating with the SPI slave device.

ni845xSpiConfigurationSetClockRate

Purpose

Sets the configuration clock rate in kilohertz.

Format

```
int32 ni845xSpiConfigurationSetClockRate (
      uInt32 ConfigurationHandle,
      uInt16 ClockRate
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`uInt16 ClockRate`

Specifies the SPI clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports. If your hardware does not support the supplied clock rate, a warning is generated, and the next smallest supported clock rate is used.
If the supplied clock rate is smaller than the smallest supported clock rate, an error is generated.

The default value for the clock rate is 1000 kHz (1 MHz).

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationSetClockRate` to set the SPI configuration clock rate in kilohertz.

ni845xSpiConfigurationSetPort

Purpose

Sets the configuration port number.

Format

```
int32 ni845xSpiConfigurationSetPort (
       uInt32 ConfigurationHandle,
       uInt8  Port
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`uInt8 Port`

Specifies the SPI port that this configuration communicates across.

Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the number of SPI ports your NI 845*x* device supports.

The default value for the port number is 0.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiConfigurationSetPort` to select the SPI port where the SPI slave resides.

Basic

ni845xSpiWriteRead

Purpose

Exchanges an array of data with an SPI slave device.

Format

```
int32 ni845xSpiWriteRead (
       uInt32   DeviceHandle,
       uInt32   ConfigurationHandle,
       uInt32   WriteSize,
       uInt8  * pWriteData,
       uInt32 * pReadSize,
       uInt8  * pReadData
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiConfigurationOpen`.

`uInt32 WriteSize`

The number of bytes to write. This must be nonzero.

`uInt8 * pWriteData`

The data bytes to be written.

Outputs

`uInt32 * pReadSize`

A pointer to the amount of bytes read.

`uInt8 * pReadData`

A pointer to an array of bytes where the bytes that have been read are stored.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiWriteRead` to exchange an array of data with an SPI slave device. Due to the full-duplex nature of SPI, the read data size equals the write data size, unless there is an error. Some SPI devices act as receivers only and require one or more command and data bytes to be sent to them in one SPI transaction. As this is device specific, you must review the device datasheet to package the required commands and data into the write data array. Other SPI devices act as transceivers. These devices can receive data much like receiver-only devices. But they can also transmit data, which usually requires writing one or more command bytes plus a number of bytes equal to the number of bytes desired to be read from the device. In most cases, the values of these bytes are not important, as they serve only to clock data out of the device. Here again, the SPI transaction formats are device specific, so you must review the device datasheet to package the required commands and data into the write data array.

Before using `ni845xSpiWriteRead`, you must ensure that the configuration parameters specified in `ConfigurationHandle` are correct for the device you currently want to access.

Advanced

ni845xSpiScriptClockPolarityPhase

Purpose

Adds an SPI Script Clock Polarity Phase command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA).

Format

```
int32 ni845xSpiScriptClockPolarityPhase (
       uInt32 ScriptHandle,
       int32  Polarity,
       int32  Phase
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`int32 Polarity`

The clock line idle state for the SPI Port. `Polarity` uses the following values:

- `kNi845xSpiClockPolarityIdleLow` (0): Clock is low in the idle state.
- `kNi845xSpiClockPolarityIdleHigh` (1): Clock is high in the idle state.

`int32 Phase`

The positioning of the data bits relative to the clock edges for the SPI Port. `Phase` uses the following values:

- `kNi845xSpiClockPhaseFirstEdge` (0): Data is centered on the first edge of the clock period.
- `kNi845xSpiClockPhaseSecondEdge` (1): Data is centered on the second edge of the clock period.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptClockPolarityPhase` to add an SPI Script Clock Polarity Phase command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA) for the SPI port you specify when you use `ni845xSpiScriptRun` to execute the script. `Polarity` sets SPI clock line idle state. The default (`kNi845xSpiClockPolarityIdleLow`) sets the clock line to idle at a low logic level. Setting the clock polarity to `kNi845xSpiClockPolarityIdleHigh` sets the clock line to idle at a high logic level. `Phase` sets the SPI clock edge on which the NI-845*x* SPI port centers each MOSI data bit. The default (`kNi845xSpiClockPhaseFirstEdge`) centers each MOSI data bit on the first edge of each clock cycle. Setting the clock phase to `kNi845xSpiClockPhaseSecondEdge` causes each MOSI data bit to be centered on the second edge of each clock cycle.

ni845xSpiScriptClockRate

Purpose

Adds an SPI Script Clock Rate command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock rate in kilohertz.

Format

```
int32 ni845xSpiScriptClockRate (
      uInt32 ScriptHandle,
      uInt16 ClockRate
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt16 ClockRate`

The SPI clock rate in kilohertz. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845*x* device supports.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptClockRate` to add an SPI Script Clock Rate command to an SPI script referenced by `ScriptHandle`. This command sets the SPI clock rate for the SPI port you specify when you use `ni845xSpiScriptRun` to execute the script. The NI 845*x* device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845*x* software adjusts it down to a supported rate and generates a warning. If the selected rate is lower than all supported rates, an error is generated.

ni845xSpiScriptClose

Purpose

Closes a previously opened script handle.

Format

```
int32 ni845xSpiScriptClose (uInt32 ScriptHandle);
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptClose` to close a previously opened reference.

ni845xSpiScriptCSHigh

Purpose

Adds an SPI Script CS High command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic high state.

Format

```
int32 ni845xSpiScriptCSHigh (
      uInt32 ScriptHandle,
      uInt32 ChipSelectNum
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 ChipSelect`

The chip select to set high.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptCSHigh` to add an SPI Script CS High command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic high state.

ni845xSpiScriptCSLow

Purpose

Adds an SPI Script CS Low command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic low state.

Format

```
int32 ni845xSpiScriptCSLow (
      uInt32 ScriptHandle,
      uInt32 ChipSelectNum
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 ChipSelect`

The chip select to set low.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptCSLow` to add an SPI Script CS Low command to an SPI script referenced by `ScriptHandle`. This command sets an SPI chip select to the logic low state.

ni845xSpiScriptDelay

Purpose

Adds an SPI Script Delay command to an SPI script referenced by `ScriptHandle`. This command adds a delay after the previous SPI script command.

Format

```
int32 ni845xSpiScriptDelay (
       uInt32 ScriptHandle,
       uInt8  Delay
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 Delay`

The desired delay in milliseconds.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDelay` to add an SPI Script Delay command to an SPI script referenced by `ScriptHandle`. This command adds a delay after the previous SPI script command.

ni845xSpiScriptDioConfigureLine

Purpose

Adds an SPI Script DIO Configure Line command to an SPI script referenced by `ScriptHandle`. This command configures a DIO line on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioConfigureLine (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  LineNumber,
       int32  ConfigurationValue
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to configure.

`int32 ConfigurationValue`

The line configuration. `ConfigurationValue` uses the following values:

- `kNi845xDioInput` (0): The line is configured for input.
- `kNi845xDioOutput` (1): The line is configured for output.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioConfigureLine` to add an SPI Script DIO Configure Line command to an SPI script referenced by `ScriptHandle`. This command allows you to configure one line, specified by `LineNumber`, of a byte-wide DIO port, as an input or output. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xSpiScriptDioConfigurePort

Purpose

Adds an SPI Script DIO Configure Port command to an SPI script referenced by `ScriptHandle`. This command configures a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioConfigurePort (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  ConfigurationValue
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port to configure.

`uInt8 ConfigurationValue`

A bitmap that specifies the function of each individual line of a port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioConfigurePort` to add an SPI Script DIO Configure Port command to an SPI script referenced by `ScriptHandle`. This command allows you to configure all eight lines of a byte-wide DIO port. Setting a bit to 1 configures the corresponding DIO port line for output. Setting a bit to 0 configures the corresponding port line for input. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the port to configure. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xSpiScriptDioReadLine

Purpose

Adds an SPI Script DIO Read Line command to an SPI script referenced by `ScriptHandle`. This command reads from a DIO line on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioReadLine(
       uInt32   ScriptHandle,
       uInt8    PortNumber,
       uInt8    LineNumber,
       uInt32 * pScriptReadIndex
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to read.

Outputs

`uInt32 * pScriptReadIndex`

An unsigned 32-bit integer pointer that stores the script read index. `pScriptReadIndex` is the index of the read command within the script. It is used as an input into `ni845xSpiScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioReadLine` to add an SPI Script DIO Read command to an SPI script referenced by `ScriptHandle`. This command allows you to read one line, specified by `LineNumber`, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the

`PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

To obtain the logic level read from the specified DIO port line, pass the value of `pScriptReadIndex` to `ni845xSpiScriptExtractReadDataSize` to retrieve the read data size and `ni845xSpiScriptExtractReadData` after script execution. `ni845xSpiScriptExtractReadData` returns either `kNi845xDioLogicLow` if the logic level read on the specified line was low or `kNi845xDioLogicHigh` if the logic level read on the specified line was high.

ni845xSpiScriptDioReadPort

Purpose

Adds an SPI Script DIO Read Port command to an SPI script referenced by `ScriptHandle`. This command reads from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioReadPort (
       uInt32   ScriptHandle,
       uInt8    PortNumber,
       uInt32 * pScriptReadIndex
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port to read.

Outputs

`uInt32 * pScriptReadIndex`

An unsigned 32-bit integer pointer that stores the script read index. `pScriptReadIndex` is the index of the read command within the script. It is used as an input into `ni845xSpiScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioReadPort` to add an SPI Script DIO Read Port command to an SPI script referenced by `ScriptHandle`. Use this command to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

To obtain the data byte read from the specified DIO port, pass the value of `pScriptReadIndex` to `ni845xSpiScriptExtractReadDataSize` to retrieve the read data size and `ni845xSpiScriptExtractReadData` after script execution, which returns the data byte read by this script command.

ni845xSpiScriptDioWriteLine

Purpose

Adds an SPI Script DIO Write Line command to an SPI script referenced by `ScriptHandle`. This command writes to a DIO line on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioWriteLine (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  LineNumber,
       int32  WriteData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to write.

`int32 WriteData`

The value to write to the line. `WriteData` uses the following values:

- `kNi845xDioLogicLow` (0): The line is set to the logic low state.
- `kNi845xDioLogicHigh` (1): The line is set to the logic high state.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioWriteLine` to add an SPI Script DIO Write Line command to an SPI script referenced by `ScriptHandle`. Use this command to write one line, specified by `LineNumber`, of a byte-wide DIO port. If `WriteData` is `kNi845xDioLogicHigh`, the specified line's output is driven to a high logic level. If `WriteData` is `kNi845xDioLogicLow`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xSpiScriptDioWritePort

Purpose

Adds an SPI Script DIO Write Port command to an SPI script referenced by `ScriptHandle`. This command writes to a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSpiScriptDioWritePort (
       uInt32 ScriptHandle,
       uInt8  PortNumber,
       uInt8  WriteData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt8 PortNumber`

The DIO port to write.

`uInt8 WriteData`

The value to write to the DIO port. Only lines configured for output are updated.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDioWritePort` to add an SPI Script DIO Write Port command to an SPI script referenced by `ScriptHandle`. Use this command to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xSpiScriptDisableSPI

Purpose

Adds an SPI Script Disable SPI command to an SPI script referenced by `ScriptHandle`. This command tristates the pins on an SPI port specified using `ni845xSpiScriptRun`. It also tristates all chip select pins.

Format

```
int32 ni845xSpiScriptDisableSPI (
       uInt32 ScriptHandle
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptDisableSPI` to add an SPI Script Disable SPI command to an SPI script referenced by `ScriptHandle`. This command tristates the pins on the SPI port you specify when you use `ni845xSpiScriptRun`. All chip select pins are also tristated.

ni845xSpiScriptEnableSPI

Purpose

Adds an SPI Script Enable SPI command to an SPI script referenced by `ScriptHandle`. This command switches the pins on an SPI port specified using `ni845xSpiScriptRun` to master mode function. All chip select pins are switched from tristate to push-pull output driven high.

Format

```
int32 ni845xSpiScriptEnableSPI (
      uInt32 ScriptHandle
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptEnableSPI` to add an SPI Script Enable SPI command to an SPI script referenced by `ScriptHandle`. This command switches the pins on the SPI port you specify when you use `ni845xSpiScriptRun`, from tristate to master mode function. Also, all chip select pins are switched from tristate to push-pull output driven high. It is important to keep this in mind if you are creating a script to access a device with an active high chip select input. You need to enable SPI and write the device chip select low until you want to access it, at which time you set the chip select high, perform the write/read, and then set the chip select low.

ni845xSpiScriptExtractReadData

Purpose

Extracts the desired read data from an SPI script, referenced by `ScriptHandle`, which has been processed by `ni845xSpiScriptRun`. Each script read command (`ni845xSpiScriptWriteRead`, `ni845xSpiScriptDioReadPort`, `ni845xSpiScriptDioReadLine`) returns a script read index. You can extract data for each script read index in a script, by passing each index to a separate call of `ni845xSpiScriptExtractReadData`.

Format

```
int32 ni845xSpiScriptExtractReadData (
       uInt32  ScriptHandle,
       uInt32  ScriptReadIndex,
       uInt8 * pReadData
       );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 ScriptReadIndex`

Identifies the read in the script whose data should be extracted.

Outputs

`uInt8 * pReadData`

The data returned for the script command specified by `ScriptReadIndex`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptExtractReadData` to extract the desired read data from an SPI script, indicated by `ScriptHandle`, which has been processed by `ni845xSpiScriptRun`. Each SPI script read command (`ni845xSpiScriptWriteRead`, `ni845xSpiScriptDioReadPort`, `ni845xSpiScriptDioReadLine`) returns a script read index.

ni845xSpiScriptExtractReadDataSize

Purpose

Retrieves the read data size from an SPI script, referenced by `ScriptHandle`, which has been processed by `ni845xSpiScriptRun`. Each script read command (`ni845xSpiScriptWriteRead`, `ni845xSpiScriptDioReadPort`, `ni845xSpiScriptDioReadLine`) returns a script read index. You can extract data for each script read index in a script, by passing each index to `ni845xSpiScriptExtractReadData`.

Format

```
int32 ni845xSpiScriptExtractReadDataSize (
      uInt32   ScriptHandle,
      uInt32   ScriptReadIndex,
      uInt32 * pReadDataSize
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 ScriptReadIndex`

Identifies the read in the script whose data size should be extracted.

Outputs

`uInt32 * pReadDataSize`

Stores the read data buffer size at the given index.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptExtractReadDataSize` to retrieve the desired read data size from an SPI script, indicated by `ScriptHandle`, which has been processed by `ni845xSpiScriptRun`. Each SPI script read command (`ni845xSpiScriptWriteRead`, `ni845xSpiScriptDioReadPort`, `ni845xSpiScriptDioReadLine`) returns a script read index.

ni845xSpiScriptOpen

Purpose

Creates a new NI-845*x* SPI script.

Format

```
int32 ni845xSpiScriptOpen (uInt32 * pScriptHandle);
```

Inputs

None.

Outputs

`uInt32 * pScriptHandle`

A pointer to an unsigned 32-bit integer to store the new script handle in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptOpen` to create a new script to use with the NI-845*x* SPI Advanced API. Pass the reference to SPI script functions to create the script. Then, call `ni845xSpiScriptRun` to execute your script on your NI 845*x* device. After you finish executing your script, use `ni845xSpiScriptClose` to delete the script.

ni845xSpiScriptReset

Purpose

Resets an SPI script referenced by `ScriptHandle` to an empty state.

Format

```
int32 ni845xSpiScriptReset (uInt32 ScriptHandle);
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptReset` to reset a script to an empty state. Any commands or read data stored in the script are deleted.

ni845xSpiScriptRun

Purpose

Sends the SPI script to the desired NI 845*x* device, which then interprets and runs it.

Format

```
int32 ni845xSpiScriptRun (
      uInt32 ScriptHandle,
      uInt32 DeviceHandle,
      uInt8  PortNumber
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

The SPI port this script runs on.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptRun` to execute an SPI script referenced by `ScriptHandle` on the device referenced by `DeviceHandle`. You must first create an SPI script using the SPI scripting functions. Next, pass the script handle into `ScriptHandle`. If you have multiple NI 845*x* devices installed in your system, you can select which device to write your SPI script to by passing its handle into `DeviceHandle`. If your NI 845*x* device supports multiple SPI ports, you can also select which port to write your SPI script to. For single SPI port NI 845*x* devices, you must use the default port (0). In this way, you can create one script to run on various NI 845*x* devices, on various SPI ports within those devices. `ni845xSpiScriptRun` loads and executes your SPI script on the NI 845*x* device and SPI port you specify, then returns success or error. If your script contained any read commands, you can use `ni845xSpiScriptExtractReadData` to extract the read data after executing `ni845xSpiScriptRun`.

ni845xSpiScriptWriteRead

Purpose

Adds an SPI Script Write Read command to an SPI script referenced by `ScriptHandle`. This command exchanges an array of data with an SPI slave device.

Format

```
int32 ni845xSpiScriptWriteRead (
      uInt32   ScriptHandle,
      uInt32   WriteSize,
      uInt8  * pWriteData,
      uInt32 * pScriptReadIndex
      );
```

Inputs

`uInt32 ScriptHandle`

The script handle returned from `ni845xSpiScriptOpen`.

`uInt32 WriteSize`

The number of bytes to write. This must be nonzero.

`uInt8 * pWriteData`

The bytes to write.

Outputs

`uInt32 * pScriptReadIndex`

A pointer to the write/read command index within the script. It is used as an input into `ni845xSpiScriptExtractReadData`.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiScriptWriteRead` to add an SPI Script Write Read command to an SPI script referenced by `ScriptHandle`. This command exchanges an array of data with an SPI slave device connected to the SPI port you specify when you use `ni845xSpiScriptRun` to execute the script. Due to the full-duplex nature of SPI, the read data size equals the write data size, unless there is an error. Some SPI devices act as receivers only and require one or more command and data bytes to be sent to them in one SPI transaction. As this is device specific, you need to review the device datasheet to package the required commands and data into the write data array. Other SPI devices act as transceivers. These devices can receive data much like receiver-only devices. But they can also transmit data, which usually requires writing one or more command bytes plus a number of bytes equal to the number of bytes desired to be read from the device. In most cases, the values of these bytes are not important, as they serve only to clock data out of the device. Here again, the SPI transaction formats are device specific, so you need to review the device datasheet to package the required commands and data into the write data array. To obtain the data read from the specified SPI port, pass the value of `pScriptReadIndex` to `ni845xSpiScriptExtractReadData` after script execution, which returns the data read by this script command.

11

Using the NI-845*x* SPI Stream API

This chapter helps you get started with the NI-845*x* SPI Stream API.

NI-845*x* SPI Stream Programming Model

The SPI Stream API provides the highest performance SPI transaction by allowing you to configure a timing waveform for SPI and DIO signals. This API is ideal for reading high-speed streaming data from an SPI slave device, such as an analog-to-digital converter (ADC).

When using the SPI Stream API, the first step is to create an SPI stream configuration to describe the streaming waveform, as shown in Figure 11-1. To make an SPI stream configuration, create an SPI stream configuration reference and set the appropriate properties. Once the configuration has the desired settings, start the streaming operation on the hardware. Your NI 845*x* device then generates the waveform that the configuration specifies onto the SPI bus and buffer data on board. To pull data from the buffer, use the API to read data. This does not interrupt SPI transactions occurring on the device. Once the desired amount of data has been read, stop the streaming operation on the device to return to normal mode.

Note Data continues to be buffered on the device until the specified number of samples are acquired or the streaming mode is stopped.

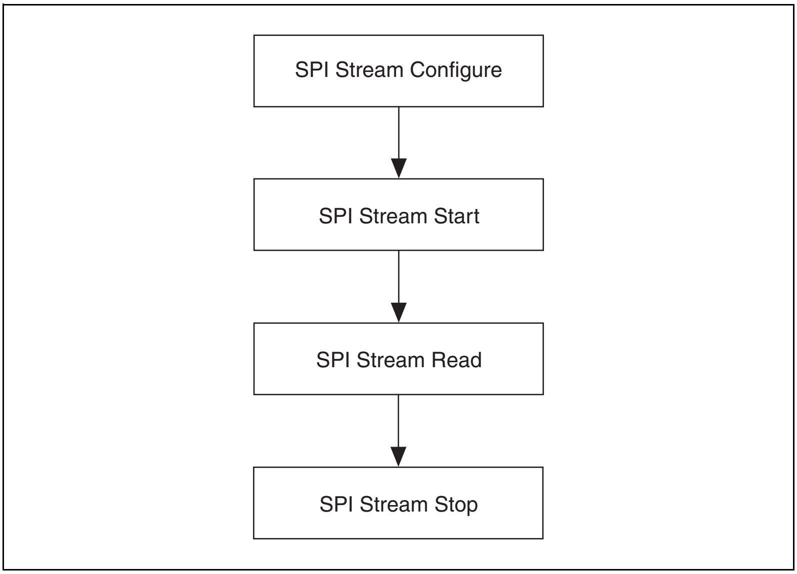


Figure 11-1. NI-845*x* SPI API Stream Programming Model

SPI Stream Configure

Use the **NI-845x SPI Stream Configuration Property Node** in LabVIEW and `ni845xSpiStreamConfiguration*` calls in other languages to set the specific SPI stream configuration that describes the characteristics of the device to communicate with.

SPI Stream Start

Use **NI-845x SPI Stream Start.vi** in LabVIEW and `ni845xSpiStreamStart` in other languages to change the device mode to streaming and start generating the specified waveform on the SPI bus.

SPI Stream Read

Use **NI-845x SPI Stream Read.vi** in LabVIEW and `ni845xSpiStreamRead` in other languages to read data from the buffer on the NI 845*x* device.

SPI Stream Stop

Use **NI-845x SPI Stream Stop.vi** in LabVIEW and `ni845xSpiStreamStop` in other languages to change the device mode to normal mode.

Waveform 1

Figure 11-2 shows the waveform 1 timing diagram. Each timing parameter is specified as a number of system clocks. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for a system clock description.

Depending on your pin configuration, not all timing parameters are used. Only the necessary timing parameters are applied when generating the waveform.

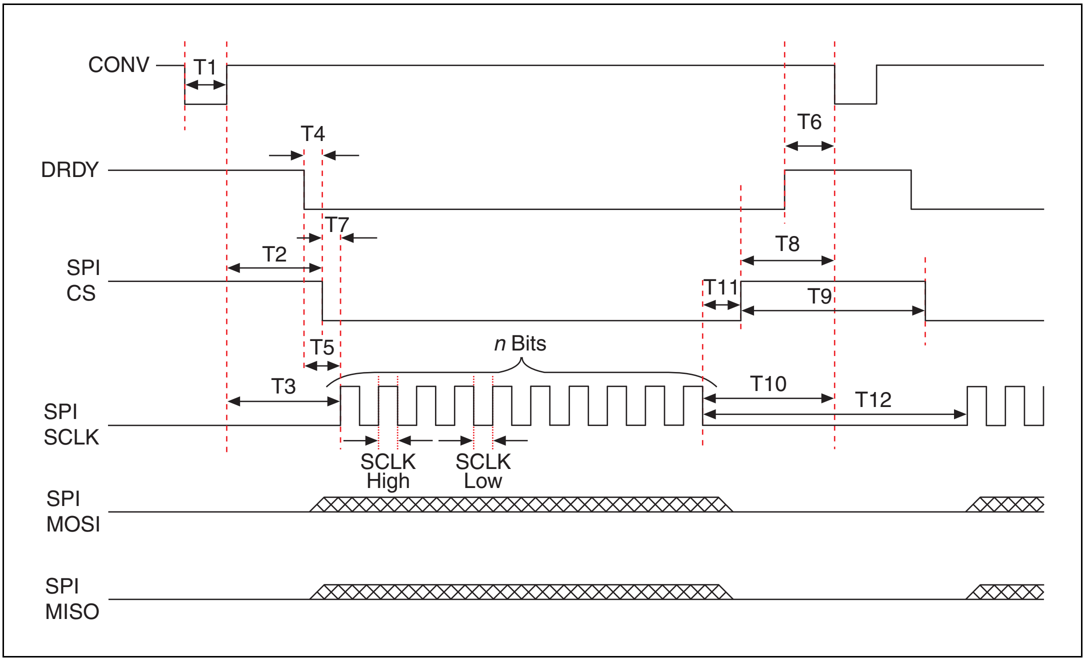


Figure 11-2. Waveform 1 Timing Diagram

Table 11-1 describes the timing parameters used depending on your pin configuration. Using the timing parameters and pin configurations, you can configure the waveform to communicate with an SPI slave.

Table 11-1. Timing Parameters

Active Pin(s)[1]	$SCLK_H$	$SCLK_L$	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
None	✓	✓	—	—	—	—	—	—	—	—	—	—	—	✓
CONV	✓	✓	✓	—	✓	—	—	—	—	—	—	✓	—	—
DRDY	✓	✓	—	—	—	—	✓	—	—	—	—	—	—	—
CS	✓	✓	—	—	—	—	—	—	✓	—	✓	—	✓	—
CONV, CS	✓	✓	✓	✓	—	—	—	—	✓	✓	—	—	✓	—
CONV, DRDY	✓	✓	✓	—	—	—	✓	✓	—	—	—	—	—	—
DRDY, CS	✓	✓	—	—	—	✓	—	—	✓	—	—	—	✓	—
CONV, DRDY, CS	✓	✓	✓	—	—	✓	—	✓	✓	—	—	—	✓	—

[1] Pins are considered active if configured as active high or active low.

Extra SPI Pin Descriptions

CONV

The CONV pin is commonly used to signal an SPI slave to begin data conversion. When the CONV pin is configured as Active High, Active Low, Drive High, or Drive Low, the pin is configured as an output using GPIO0.

DRDY

The DRDY pin is commonly used to signal your NI 845*x* device that data is ready to be read. When the DRDY pin is configured as Active High or Active Low, the pin is configured as an input using GPIO1.

Chip Select

The Chip Select (CS) pin is commonly used to signal an SPI slave that your NI 845*x* device is intending to communicate with it. When the CS pin is configured as Active High, Active Low, Drive High, or Drive Low, the pin is configured as an output using CS0.

Note Refer to Appendix A, *NI USB-845x Hardware Specifications*, for the pinout of your NI 845*x* device.

12

NI-845*x* SPI Stream API for LabVIEW

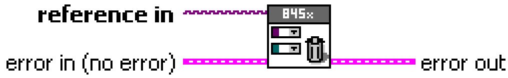
This chapter lists the LabVIEW VIs for the NI-845*x* SPI Stream API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

General Device

NI-845x Close Reference.vi

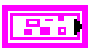
Purpose

Closes a previously opened reference.



Inputs

reference in is a reference to an NI 845*x* device, I²C configuration, SPI configuration, SPI stream configuration, I²C script, or SPI script.



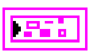
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

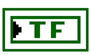
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

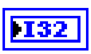
Outputs



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x Close Reference.vi** to close a previously opened reference.

NI-845x Device Property Node

Purpose

A property node with the NI-845*x* Device class preselected. This property node allows you to modify properties of your NI 845*x* device.

Inputs

device reference in is a reference to an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to an NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x Device Property Node**.

DIO:Active Port

The **DIO:Active Port** property sets the active DIO port for further DIO port configuration. The format for this property is a decimal string. For example, the string `0` represents DIO Port 0. The default value of this property is `0`. For NI 845*x* devices with one DIO port, the port value must be `0`.

DIO:Driver Type

The **DIO:Driver Type** property configures the active DIO port with the desired driver type characteristics. **DIO:Driver Type** uses the following values:

`Open-Drain`

The DIO driver type is configured for open-drain.

`Push-Pull`

The DIO driver type is configured for push-pull. The actual voltage driven (when sourcing a high value) is determined by the *I/O Voltage Level* property.

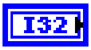
The default value of this property is `Push-Pull`.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available driver types on your hardware.

DIO:Line Direction Map

The **DIO:Line Direction Map** property sets the line direction map for the active DIO Port. The value is a bitmap that specifies the function of each individual line within the port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

The default value of this property is `0` (all lines configured for input).



I/O Voltage Level

The **I/O Voltage Level** property sets the board voltage. This property sets the voltage for SPI, I^{2}C, and DIO. The default value for this property is `3.3V`. This property uses the following values:

`3.3V`

I/O Voltage is set to 3.3 V.

`2.5V`

I/O Voltage is set to 2.5 V.

`1.8V`

I/O Voltage is set to 1.8 V.

`1.5V`

I/O Voltage is set to 1.5 V.

`1.2V`

I/O Voltage is set to 1.2 V.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

I^{2}C Pullup Enable

The **I^{2}C Pullup Enable** property enables or disables the internal pullup resistors connected to SDA and SCL.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your hardware has onboard pull-up resistors.

NI-845x Device Reference

Purpose

Specifies the device resource to be used for communication.

Description

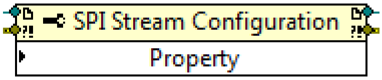
Use the **NI-845x Device Reference** to describe the NI 845*x* device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845*x* API call to invoke the function on the associated NI 845*x* device.

Configuration

NI-845x SPI Stream Configuration Property Node

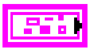
Purpose

A property node with the NI-845*x* SPI Stream Configuration class preselected. This property node allows you to query and modify SPI Stream configuration properties.



Inputs

spi stream configuration in is a reference to a specific SPI stream configuration that describes the waveform to generate during streaming operations.



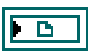
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

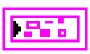
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs



spi stream configuration out is a reference to a specific SPI stream configuration that describes the waveform to generate during streaming operations.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x SPI Stream Configuration Property Node**.

Number of Samples

Sets the number of samples to acquire. For continuous streaming, this property should be set to 0.

The default value for this property is `0` (continuous streaming).

Number of Bits Per Sample

Sets the number of bits to be clocked in per sample. Refer to Chapter 3, *NI USB-845x Hardware Overview*, for valid settings for this property.

The default value for this property is `8`.

Clock Polarity

Sets the idle state of the clock line during SPI Streaming. **Clock Polarity** uses the following values:

`0` (Idle Low)

Clock is low in the idle state.

`1` (Idle High)

Clock is high in the idle state.

The default value for this property is `0` (Idle Low).

Clock Phase

Sets the positioning of the data bits relative to the clock during SPI Streaming. **Clock Phase** uses the following values:

`0` (First Edge)

Data is centered on the first edge of the clock period.

`1` (Second Edge)

Data is centered on the second edge of the clock period.

The default value for this property is `0` (First Edge).

Packet Size

Sets the packet size for transfers between the host and your NI 845*x* device.

For most applications, set this parameter to a multiple of 512 bytes for optimal performance.

This setting can affect the performance of data streaming to the host from your NI 845*x* device. For slow SPI streaming configurations, setting this property below 512 allows data to transfer to the host more often. Setting the packet size too small, however, may cause the onboard buffer to overflow for high-speed SPI streaming operations.

Waveform1:MOSI Data

Sets the data to be used to transfer on MOSI during an SPI operation. The Number of Bits Per Sample determines the number of bytes used from the array. During an SPI sample, only the least significant bits necessary are transferred.

Note If not enough bytes are specified in the MOSI Data array, data bytes of 0 are padded to the end of the array.

Waveform1:Timing:SCLKLow

Sets the number of system clocks for the SCLK low period for Waveform 1.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:SCLKHigh

Sets the number of system clocks for the SCLK high period for Waveform 1.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T1(convA->convD)

Sets the number of system clocks between CONV assert and CONV deassert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T2(convD->csA)

Sets the number of system clocks between CONV deassert and Chip Select assert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T3(convD->sclkA)

Sets the number of system clocks between CONV deassert and SCLK assert (first bit) for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T4(drdyA->csA)

Sets the number of system clocks between DRDY assert and Chip Select assert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `2`.

Waveform1:Timing:T5(drdyA->sclkA)

Sets the number of system clocks between DRDY assert and SCLK assert (first bit) for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `2`.

Waveform1:Timing:T6(drdyD->convA)

Sets the number of system clocks between DRDY deassert and CONV assert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `2`.

Waveform1:Timing:T7(csA->sclkA)

Sets the number of system clocks between Chip Select assert and SCLK assert (first bit) for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T8(csD->convA)

Sets the number of system clocks between Chip Select deassert and CONV assert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *NI USB-845x Hardware Specifications*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T9(csD->csA)

Sets the number of system clocks between Chip Select deassert and Chip Select assert (first bit) for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T10(sclkD->convA)

Sets the number of system clocks between SCLK deassert (last bit) and CONV assert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T11(sclkD->csD)

Sets the number of system clocks between SCLK deassert (last bit) and CS deassert for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Timing:T12(sclkD->sclkA)

Sets the number of system clocks between SCLK deassert (last bit) and SCLK assert (first bit) for Waveform 1. Depending on pin settings, this timing parameter may not be applicable. The value stored in this setting is ignored at runtime if the timing parameter is not necessary. Refer to Figure 11-2, *Waveform 1 Timing Diagram*, in Chapter 11, *Using the*

NI-845x SPI Stream API, to determine the timing parameters used for your application.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid values for this parameter. The default value is `1`.

Waveform1:Pin:CONV

Sets the configuration for the CONV pin. **Waveform1:Pin:CONV** uses the following values:

`Disabled`

The pin is disabled.

`Active High`

The pin is set to active high.

`Active Low`

The pin is set to active low.

`Drive High`

The pin is driven high.

`Drive Low`

The pin is driven low.

Waveform1:Pin:DRDY

Sets the configuration for the DRDY pin. **Waveform1:Pin:DRDY** uses the following values:

`Disabled`

The pin is disabled.

`Active High`

The pin is set to active high.

`Active Low`

The pin is set to active low.

Waveform1:Pin:CS

Sets the configuration for the Chip Select pin. **Waveform1:Pin:CS** uses the following values:

`Disabled`

The pin is disabled.

`Active High`

The pin is set to active high.

`Active Low`

The pin is set to active low.

`Drive High`

The pin is driven high.

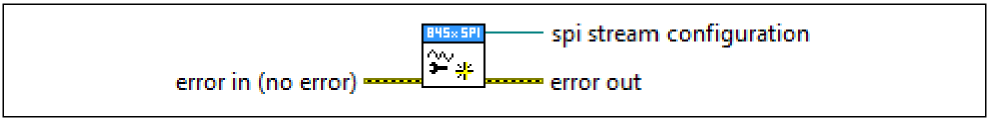
`Drive Low`

The pin is driven low.

NI-845x SPI Stream Create Configuration Reference.vi

Purpose

Creates a new NI-845*x* SPI Stream configuration.



Inputs

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

spi stream configuration is a reference to the newly created NI-845*x* SPI stream configuration.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

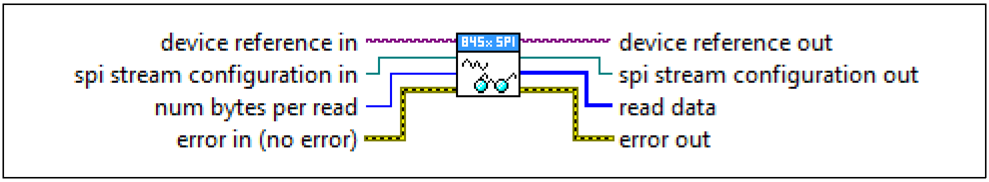
Use **NI-845x SPI Stream Create Configuration Reference.vi** to create a new configuration to use with the NI-845*x* SPI Stream API. Pass the reference to a property node to make the configuration match the settings of your SPI slave. Then, pass the configuration to the SPI stream functions to execute them on the described SPI slave. After you finish communicating with your SPI slave, pass the reference into a new property node to reconfigure it or use **NI-845x Close Reference.vi** to delete the configuration.

Basic

NI-845x SPI Stream Read.vi

Purpose

Reads data from an SPI slave device

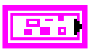


Inputs

device reference in is a reference to an NI 845*x* device.

spi stream configuration in is a reference to a specific SPI stream configuration that describes the waveform to generate during streaming operations. Connect this configuration reference to a property node to set the specific configuration parameters.

num bytes per read contains the number of bytes to attempt to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

spi stream configuration out is a reference to the SPI stream configuration after this VI runs.

read data contains an array of read data from an SPI interface. All data is padded to the nearest byte with zeros as the most significant bits.

Note A pad byte of 0 may be added to the end of a finite acquisition if the total number of bytes read from the NI 845*x* device is not even.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

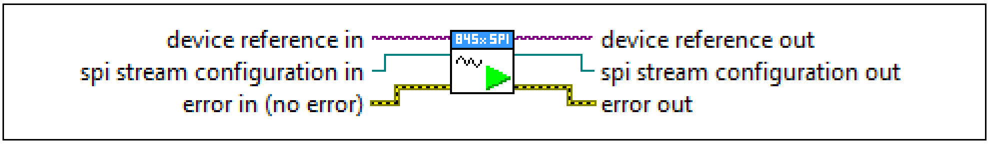
Use **NI-845x SPI Stream Read.vi** to read data from an SPI slave device. The read size is less than or equal to the value passed into **num bytes per read** and is dependent on the Packet Size.

While your NI 845*x* device is in streaming mode, SPI operations continue to occur and buffer onboard. **NI-845x SPI Stream Read.vi** does not affect SPI operations on the SPI bus. This function reads the result of the started SPI streaming operation.

NI-845x SPI Stream Start.vi

Purpose

Starts the streaming operation on an NI 845*x* device.



Inputs

device reference in is a reference to an NI 845*x* device.

spi stream configuration in is a reference to a specific SPI stream configuration that describes the waveform to generate during streaming operations. Connect this configuration reference to a property node to set the specific configuration parameters.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

spi stream configuration out is a reference to the SPI stream configuration after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x SPI Stream Start.vi** to put your NI 845*x* device into streaming mode. Once in streaming mode, your NI 845*x* device generates the waveform described by **spi stream configuration in**. Your NI 845*x* device remains in streaming mode until **NI-845x SPI Stream Stop.vi** is called.

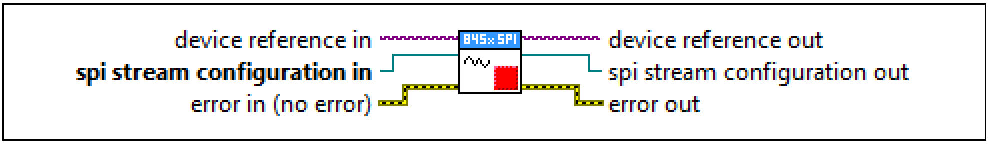
The data set in **Waveform1:MOSI Data** is output on MOSI on each SPI operation during streaming. You can use this data to set up the SPI slave if necessary, but not all SPI slaves require it.

Before using **NI-845x SPI Stream Start.vi**, you must ensure that the configuration parameters specified in **spi stream configuration in** are correct for the device you currently want to access.

NI-845x SPI Stream Stop.vi

Purpose

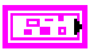
Stops a streaming operation on an NI 845*x* device.



Inputs

device reference in is a reference to an NI 845*x* device.

spi stream configuration in is a reference to a specific SPI stream configuration that describes the waveform to generate during streaming operations. Connect this configuration reference to a property node to set the specific configuration parameters.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

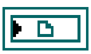
status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

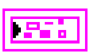
source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.



spi stream configuration out is a reference to the SPI stream configuration after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x SPI Stream Stop.vi** to remove your NI 845*x* device from streaming mode. When stopping, the device waits for the final SPI operation to complete if one is occurring. No data can be read from the device once stopped. All unread data is discarded.

13

NI-845*x* SPI Stream API for C

This chapter lists the functions for the NI-845*x* SPI Stream API for C and describes the format, purpose, and parameters for each function. The functions are listed alphabetically in four categories: general device, configuration, basic, and advanced.

Section Headings

The NI-845*x* SPI Stream API for C functions include the following section headings.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Inputs and Outputs

These sections list the function input and output parameters.

Description

The description section gives details about the purpose and effect of each function.

Data Types

The NI-845*x* SPI Stream API for C functions use the following data types.

Data Type	Purpose
`uInt8`	8-bit unsigned integer
`uInt16`	16-bit unsigned integer
`uInt32`	32-bit unsigned integer
`int8`	8-bit signed integer

Data Type	Purpose
`int16`	16-bit signed integer
`int32`	32-bit signed integer
`uInt8 *`	Pointer to an 8-bit unsigned integer
`uInt16 *`	Pointer to a 16-bit unsigned integer
`uInt32 *`	Pointer to a 32-bit unsigned integer
`int8 *`	Pointer to an 8-bit signed integer
`int16 *`	Pointer to a 16-bit signed integer
`int32 *`	Pointer to a 32-bit signed integer
`char *`	ASCII string represented as an array of characters terminated by null character (`'\0'`)

List of Functions

The following table contains an alphabetical list of the NI-845*x* SPI Stream API for C functions.

Function	Purpose
`ni845xClose`	Closes a previously opened NI 845*x* device.
`ni845xCloseFindDeviceHandle`	Closes the handles created by `ni845xFindDevice`.
`ni845xDeviceLock`	Locks NI 845*x* devices for access by a single thread.
`ni845xDeviceUnlock`	Unlocks NI 845*x* devices.
`ni845xFindDevice`	Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.
`ni845xFindDeviceNext`	Finds subsequent devices after `ni845xFindDevice` has been called.

Function	Purpose
ni845xOpen	Opens an NI 845*x* device for use with various write, read, and device property functions.
ni845xSpiStreamConfigurationClose	Closes an NI-845*x* SPI Stream Configuration.
ni845xSpiStreamConfigurationOpen	Creates a new NI-845*x* SPI Stream Configuration.
ni845xSpiStreamConfigurationGetNumBits	Retrieves the configuration's number of bits per sample.
ni845xSpiStreamConfigurationGetNumSamples	Retrieves the configuration's number of samples to acquire.
ni845xSpiStreamConfigurationGetPacketSize	Retrieves the configuration's packet size.
ni845xSpiStreamConfigurationGetClockPhase	Retrieves the configuration's clock phase.
ni845xSpiStreamConfigurationWave1GetPinConfig	Retrieves the configuration's setting for an individual pin.
ni845xSpiStreamConfigurationGetClockPolarity	Retrieves the configuration's clock polarity.
ni845xSpiStreamConfigurationWave1GetTimingParam	Retrieves the configuration's setting for an individual timing parameter.
ni845xSpiStreamRead	Reads data from the NI 845*x* device.
ni845xSpiStreamConfigurationWave1SetMosiData	Sets the configuration's lower 32 bits of data to be transferred on MOSI.
ni845xSpiStreamConfigurationSetNumBits	Sets the configuration's number of bits to be transferred.
ni845xSpiStreamConfigurationSetNumSamples	Sets the configuration's number of samples to be transferred.
ni845xSpiStreamConfigurationSetPacketSize	Sets the configuration's packet size.
ni845xSpiStreamConfigurationSetClockPhase	Sets the configuration's clock phase.
ni845xSpiStreamConfigurationWave1SetPinConfig	Sets the configuration's setting for an individual pin.

Function	Purpose
ni845xSpiStreamConfigurationSetClockPolarity	Sets the configuration's clock polarity.
ni845xSpiStreamConfigurationWave1SetTimingParam	Sets the configuration's setting for an individual timing parameter.
ni845xSpiStreamStart	Starts the streaming operation.
ni845xSpiStreamStop	Stops the streaming operation.
ni845xStatusToString	Converts a status code into a descriptive string.

General Device

ni845xClose

Purpose

Closes a previously opened NI 845*x* device.

Format

```
int32 ni845xClose(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be closed.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xClose` to close a device handle previously opened by `ni845xOpen`. Passing an invalid handle to `ni845xClose` is ignored.

ni845xCloseFindDeviceHandle

Purpose

Closes the handles created by `ni845xFindDevice`.

Format

```
int32 ni845xCloseFindDeviceHandle (
       uInt32 FindDeviceHandle
       );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xCloseFindDeviceHandle` to close a find list. In this process, all allocated data structures are freed.

ni845xDeviceLock

Purpose

Locks NI 845*x* devices for access by a single thread.

Format

```
int32 ni845xDeviceLock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be locked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

This function locks NI 845*x* devices and prevents multiple processes or threads from accessing the device until the process or thread that owns the device lock calls an equal number of `ni845xDeviceUnlock` calls. Any thread or process that attempts to call `ni845xDeviceLock` when the device is already locked is forced to sleep by the operating system. This is useful for when multiple Basic API device accesses must occur uninterrupted by any other processes or threads. If a thread exits without fully unlocking the device, the device is unlocked. If a thread is the current owner of the lock, and calls `ni845xDeviceLock` again, the thread will not deadlock itself, but care must be taken to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` called. This function can possibly lock a device indefinitely: If a thread never calls `ni845xDeviceUnlock`, or fails to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` call, and never exits, other processes and threads are forced to wait. This is *not* recommended for users unfamiliar with threads or processes. A simpler alternative is to use scripts. Scripts provide the same capability to ensure transfers are uninterrupted, and with possible performance benefits.

ni845xDeviceUnlock

Purpose

Unlocks NI 845*x* devices.

Format

```
int32 ni845xDeviceUnlock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be unlocked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDeviceUnlock` to unlock access to an NI 845*x* device previously locked with `ni845xDeviceLock`. Every call to `ni845xDeviceLock` must have a corresponding call to `ni845xDeviceUnlock`. Refer to `ni845xDeviceLock` for more details regarding how to use device locks.

ni845xFindDevice

Purpose

Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.

Format

```
int32 ni845xFindDevice (
      char   * FirstDevice,
      uInt32 * FindDeviceHandle,
      uInt32 * NumberFound
      );
```

Inputs

None.

Outputs

`char * FirstDevice`

A pointer to the string containing the first NI 845*x* device found. You can pass this name to the `ni845xOpen` function to open the device. If no devices exist, this is an empty string.

`uInt32 * FindDeviceHandle`

Returns a handle identifying this search session. This handle is used as an input in `ni845xFindDeviceNext` and `ni845xCloseFindDeviceHandle`.

`uInt32 * NumberFound`

A pointer to the total number of NI 845*x* devices found in the system. You can use this number in conjunction with the `ni845xFindDeviceNext` function to find a particular device. If no devices exist, this returns 0.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDevice` to get a single NI 845*x* device and the number of NI 845*x* devices in the system. You can then pass the string returned to `ni845xOpen` to access the device. If you must discover more devices, use `ni845xFindDeviceNext` with `FindDeviceHandle` and NumberFound to find the remaining NI 845*x* devices in the system. After finding all desired devices, call ni845xCloseFindDeviceHandle to close the device handle and relinquish allocated resources.

Note FirstDevice must be at least 256 bytes.

Note FindDeviceHandle and NumberFound are optional parameters. If only the first match is important, and the total number of matches is not needed, you can pass in a NULL pointer for both of these parameters, and the NI-845x driver automatically calls ni845xCloseFindDeviceHandle before this function returns.

ni845xFindDeviceNext

Purpose

Finds subsequent devices after `ni845xFindDevice` has been called.

Format

```
int32 ni845xFindDeviceNext (
      uInt32 FindDeviceHandle,
      char * NextDevice
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

`char * NextDevice`

A pointer to the string containing the next NI 845*x* device found. This is empty if no further devices are left.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDeviceNext` after first calling `ni845xFindDevice` to find the remaining devices in the system. You can then pass the string returned to `ni845xOpen` to access the device.

Note `NextDevice` must be at least 256 bytes.

ni845xOpen

Purpose

Opens an NI 845*x* device for use with various write, read, and device property functions.

Format

```
int32 ni845xOpen (
      char   * ResourceName,
      uInt32 * DeviceHandle
      );
```

Inputs

`char * ResourceName`

A resource name string corresponding to the NI 845*x* device to be opened.

Outputs

`uInt32 * DeviceHandle`

A pointer to the device handle.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xOpen` to open an NI 845*x* device for access. The string passed to `ni845xOpen` can be any of the following: an `ni845xFindDevice` device string, an `ni845xFindDeviceNext` device string, a Measurement & Automation Explorer resource name, or a Measurement & Automation Explorer alias.

ni845xStatusToString

Purpose

Converts a status code into a descriptive string.

Format

```
void ni845xStatusToString (
       int32  StatusCode,
       uInt32 MaxSize,
       int8 * StatusString
       );
```

Inputs

`int32 StatusCode`

Status code returned from an NI-845*x* function.

`uInt32 MaxSize`

Size of the `StatusString` buffer (in bytes).

Outputs

`int8 * StatusString`

ASCII string that describes `StatusCode`.

Description

When the status code returned from an NI-845*x* function is nonzero, an error or warning is indicated. This function obtains a description of the error/warning for debugging purposes.

The return code is passed into the `StatusCode` parameter. The `MaxSize` parameter indicates the number of bytes available in `StatusString` for the description (including the NULL character). The description is truncated to size `MaxSize` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `String` is null-terminated, so you can use it with ANSI C functions such as `printf`.

For applications written in C or C++, each NI-845*x* function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845*x* use of this status.

NI-845*x* Status Codes

Status Code	Meaning
Negative	Error—Function did not perform expected behavior.
Positive	Warning—Function executed, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every NI-845*x* function. If an error is detected, you should close all NI-845*x* handles, then exit the application. If a warning is detected, you can display a message for debugging purposes, or simply ignore the warning.

In some situations, you may want to check for specific errors in the code and continue communication when they occur. For example, when communicating to an I^2C EEPROM, you may expect the device to NAK its address during a write cycle, and you may use this knowledge to poll for when the write cycle has completed.

SPI Stream Configuration

ni845xSpiStreamConfigurationClose

Purpose

Closes a previously opened SPI stream configuration.

Format

```
int32 ni845xSpiStreamConfigurationClose (
      uInt32 ConfigurationHandle
      );
```

Inputs

`uInt32 ConfigurationHandle`

The SPI stream configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationClose` to close a previously opened SPI stream configuration handle. Invalid SPI stream configuration handles are ignored.

ni845xSpiStreamConfigurationOpen

Purpose

Creates a new NI-845*x* SPI stream configuration.

Format

```
int32 ni845xSpiStreamConfigurationOpen (
      uInt32 * ConfigurationHandle
      );
```

Inputs

None.

Outputs

`uInt32 * ConfigurationHandle`

A pointer to an unsigned 32-bit integer to store the configuration handle in.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationOpen` to create a new configuration to use with the NI-845*x* SPI Stream API. Pass the configuration handle to the `ni845xSpiConfigurationSet*` series of functions to make the configuration match the settings of your SPI slave. Then, pass the configuration handle to the SPI stream functions to execute them on the described SPI slave. After you finish communicating with your SPI slave, pass the configuration handle to the `ni845xSpiStreamConfigurationSet*` series of functions to reconfigure it or use `ni845xSpiStreamConfigurationClose` to delete the configuration.

ni845xSpiStreamConfigurationGetNumBits

Purpose

Retrieves the configuration's number of bits per sample.

Format

```
int32 ni845xSpiStreamConfigurationGetNumBits (
       uInt32   ConfigurationHandle,
       uInt8  * NumBits
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

`uInt8 * NumBits`

A pointer to an unsigned 8-bit integer to store the number of bits per sample.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationGetNumBits` to retrieve the number of bits per sample.

ni845xSpiStreamConfigurationGetNumSamples

Purpose

Retrieves the configuration's number of samples to acquire.

Format

```
int32 ni845xSpiStreamConfigurationGetNumSamples (
      uInt32   ConfigurationHandle,
      uInt32 * NumSamples
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

`uInt32 * NumSamples`

A pointer to an unsigned 32-bit integer to store the number of samples to stream.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationGetNumSamples` to retrieve the number of samples to stream.

ni845xSpiStreamConfigurationGetPacketSize

Purpose

Retrieves the configuration's packet size.

Format

```
int32 ni845xSpiStreamConfigurationGetPacketSize (
       uInt32    ConfigurationHandle,
       uInt32  * PacketSize
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

`uInt32 * PacketSize`

A pointer to an unsigned 32-bit integer to store the configuration's packet size.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationGetPacketSize` to retrieve the package size between the host and your NI 845*x* device.

ni845xSpiStreamConfigurationGetClockPhase

Purpose

Retrieves the configuration's clock phase.

Format

```
int32 ni845xSpiStreamConfigurationGetClockPhase (
       uInt32    ConfigurationHandle,
       uInt8   * ClockPhase
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

`uInt32 * ClockPhase`

A pointer to an unsigned 8-bit integer to store the clock phase uses the following values:

- `kNi845xSpiStreamClockPhaseFirstEdge` (0): Data is updated on the first edge of the clock period.
- `kNi845xSpiStreamClockPhaseSecondEdge` (1): Data is updated on the second edge of the clock period.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationGetClockPhase` to retrieve the clock phase used by `ConfigurationHandle`.

ni845xSpiStreamConfigurationWave1GetPinConfig

Purpose

Retrieves the configuration's setting for an individual pin.

Format

```
int32 ni845xSpiStreamConfigurationWave1GetPinConfig (
       uInt32  ConfigurationHandle,
       uInt8   PinNumber,
       uInt8 * Mode
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8 PinNumber`

An unsigned 8-bit integer to determine the pin uses the following values:

- `kNi845xSpiStreamWave1ConvPin` (0): CONV pin for Waveform 1.
- `kNi845xSpiStreamWave1DrdyPin` (1): DRDY pin for Waveform 1.
- `kNi845xSpiStreamWave1CsPin` (2): Chip Select pin for Waveform 1.

Outputs

`uInt8 * Mode`

A pointer to an 8-bit unsigned integer to store the pin mode that uses the following values:

- `kNi845xSpiStreamDisabled` (0): Pin is disabled.
- `kNi845xSpiStreamActiveHigh` (1): Pin is set to active high.
- `kNi845xSpiStreamActiveLow` (2): Pin is set to active low.
- `kNi845xSpiStreamDriveHigh` (3): Pin driven high.
- `kNi845xSpiStreamDriveLow` (4): Pin driven low.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationWave1GetPinConfig` to retrieve the configuration setting for a specific pin.

ni845xSpiStreamConfigurationGetClockPolarity

Purpose

Retrieves the configuration's clock polarity.

Format

```
int32 ni845xSpiStreamConfigurationGetClockPolarity (
       uInt32    ConfigurationHandle,
       uInt8   * ClockPolarity
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

`uInt32 * ClockPolarity`

A pointer to an unsigned 8-bit integer to store the clock phase uses the following values:

- `kNi845xSpiStreamClockPolarityIdleLow` (0): Clock is low in the idle state.
- `kNi845xSpiStreamClockPolarityIdleHigh` (1): Clock is high in the idle state.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationGetClockPolarity` to retrieve the clock polarity used by `ConfigurationHandle`.

ni845xSpiStreamConfigurationWave1GetTimingParam

Purpose

Retrieves the configuration's setting for an individual timing parameter.

Format

```
int32 ni845xSpiStreamConfigurationWave1GetTimingParam (
       uInt32   ConfigurationHandle,
       uInt8    TimingParameter,
       uInt32 * ParameterValue
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8 TimingParameter`

An unsigned 8-bit integer to determine the timing parameter uses the following values:

- `kNi845xSpiStreamWave1SclkL` (0): SCLK low period for Waveform 1.
- `kNi845xSpiStreamWave1SclkH` (1): SCLK high period for Waveform 1.
- `kNi845xSpiStreamWave1T1` (2): Timing Parameter T1—CONV assert to CONV deassert for Waveform 1.
- `kNi845xSpiStreamWave1T2` (3): Timing Parameter T2—CONV deassert to Chip Select assert for Waveform 1.
- `kNi845xSpiStreamWave1T3` (4): Timing Parameter T3—CONV deassert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T4` (5): Timing Parameter T4—DRDY assert to Chip Select assert for Waveform 1.
- `kNi845xSpiStreamWave1T5` (6): Timing Parameter T5—DRDY assert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T6` (7): Timing Parameter T6—DRDY deassert to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T7` (8): Timing Parameter T7—Chip Select assert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T8` (9): Timing Parameter T8—Chip Select deassert to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T9` (10): Timing Parameter T9—Chip Select deassert to Chip Select assert.

- `kNi845xSpiStreamWave1T10` (11): Timing Parameter T10—SCLK deassert (last bit) to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T11` (12): Timing Parameter T11—SCLK deassert (last bit) to Chip Select deassert for Waveform 1.
- `kNi845xSpiStreamWave1T12` (13): Timing Parameter T12—SCLK deassert (last bit) to SCLK assert (first bit) for Waveform 1.

Outputs

`uInt32 * ParameterValue`

A pointer to an 32-bit unsigned integer to store the timing parameter in system clocks.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationWave1GetTimingParam` to retrieve a specific timing parameter. Timing parameters are returned as number of system clocks. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for a description of the system clock on your NI 845*x* device.

ni845xSpiStreamConfigurationWave1SetMosiData

Purpose

Sets the configuration MOSI data.

Format

```
int32 ni845xSpiStreamConfigurationWave1SetMosiData (
       uInt32  ConfigurationHandle,
       uInt8 * DataArray,
       uInt32  ArraySize
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8* DataArray`

An array of unsigned 8-bit integers used to specify the data transferred on MOSI.

`uInt32 ArraySize`

Size of `DataArray` supplied.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationWave1SetMosiData` to set the data for transferring on MOSI during an SPI operation. The number of bits per sample determines the number of bytes used from the array. During an SPI sample, only the least significant bits necessary are transferred.

Note If not enough bytes are specified in the MOSI data array, data bytes of 0 are padded to the end of the array.

ni845xSpiStreamConfigurationSetNumBits

Purpose

Sets the configuration's number of bits per sample.

Format

```
int32 ni845xSpiStreamConfigurationSetNumBits (
       uInt32  ConfigurationHandle,
       uInt8   NumBits
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8 NumBits`

An unsigned 8-bit integer that contains the number of bits per sample.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationSetNumBits` to set the number of bits per sample. Each SPI operation uses the number of bits this function specifies. The default for this setting is 8-bit transfers. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid settings for this property.

ni845xSpiStreamConfigurationSetNumSamples

Purpose

Sets the configuration's number of samples to acquire.

Format

```
int32 ni845xSpiStreamConfigurationSetNumSamples (
       uInt32  ConfigurationHandle,
       uInt32  NumSamples
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt32 NumSamples`

An unsigned 32-bit integer to set the number of samples to stream.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationSetNumSamples` to set the number of samples to stream. Setting this parameter to 0 indicates infinite streaming. If this parameter is nonzero, the NI 845*x* device automatically stops streaming after the specified number of samples have been transferred.

ni845xSpiStreamConfigurationSetPacketSize

Purpose

Sets the configuration's packet size.

Format

```
int32 ni845xSpiStreamConfigurationSetPacketSize (
      uInt32 ConfigurationHandle,
      uInt32 PacketSize
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt32 PacketSize`

An unsigned 32-bit integer to set the packet size.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationSetPacketSize` to configure the packet size between the host and your NI 845*x* device.

For most applications, this parameter should be set to a multiple of 512 bytes for optimal performance. This setting can affect the performance of data streaming to the host from your NI 845*x* device. For slow SPI streaming configurations, this setting allows data to transfer to the host more often. Setting the packet size too small may cause the onboard buffer to overflow for high-speed SPI streaming operations.

ni845xSpiStreamConfigurationSetClockPhase

Purpose

Sets the configuration's clock phase.

Format

```
int32 ni845xSpiStreamConfigurationSetClockPhase (
       uInt32  ConfigurationHandle,
       uInt8   ClockPhase
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt32 ClockPhase`

An unsigned 8-bit integer to store the clock phase uses the following values:

- `kNi845xSpiStreamClockPhaseFirstEdge` (0): Data is updated on the first edge of the clock period.
- `kNi845xSpiStreamClockPhaseSecondEdge` (1): Data is updated on the second edge of the clock period.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationSetClockPhase` to set clock phase used by `ConfigurationHandle` when communicating with an SPI slave device.

ni845xSpiStreamConfigurationWave1SetPinConfig

Purpose

Sets the configuration's setting for an individual pin.

Format

```
int32 ni845xSpiStreamConfigurationWave1SetPinConfig (
      uInt32  ConfigurationHandle,
      uInt8   PinNumber,
      uInt8   Mode
      );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8 PinNumber`

An unsigned 8-bit integer to determine the pin uses the following values:

- `kNi845xSpiStreamWave1ConvPin` (0): CONV output for Waveform 1.
- `kNi845xSpiStreamWave1DrdyPin` (1): DRDY input for Waveform 1.
- `kNi845xSpiStreamWave1CsPin` (2): Chip Select output for Waveform 1.

`uInt8 Mode`

An 8-bit unsigned integer to set the pin mode that uses the following values:

- `kNi845xSpiStreamDisabled` (0): Pin is disabled.
- `kNi845xSpiStreamActiveHigh` (1): Pin is set to active high.
- `kNi845xSpiStreamActiveLow` (2): Pin is set to active low.
- `kNi845xSpiStreamDriveHigh` (3): Pin driven high.
- `kNi845xSpiStreamDriveLow` (4): Pin driven low.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationWave1SetPinConfig` to set the configuration for a specific pin. If a pin is described as an output, all modes are available. If a pin is described as an input, `kNi845xSpiStreamDriveHigh` and `kNi845xSpiStreamDriveHigh` cannot be used.

ni845xSpiStreamConfigurationSetClockPolarity

Purpose

Sets the configuration's clock polarity.

Format

```
int32 ni845xSpiStreamConfigurationSetClockPolarity (
       uInt32  ConfigurationHandle,
       uInt8   ClockPolarity
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt32 ClockPolarity`

An unsigned 8-bit integer to set the clock phase uses the following values:

- `kNi845xSpiStreamClockPolarityIdleLow` (0): Clock is low in the idle state.
- `kNi845xSpiStreamClockPolarityIdleHigh` (1): Clock is high in the idle state.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationSetClockPolarity` to set the clock polarity used by `ConfigurationHandle` when communicating with an SPI slave device.

ni845xSpiStreamConfigurationWave1SetTimingParam

Purpose

Retrieves the configuration's setting for an individual timing parameter.

Format

```
int32 ni845xSpiStreamConfigurationWave1SetTimingParam (
       uInt32  ConfigurationHandle,
       uInt8   TimingParameter,
       uInt32  ParameterValue
       );
```

Inputs

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt8 TimingParameter`

An unsigned 8-bit integer to determine the timing parameter uses the following values:

- `kNi845xSpiStreamWave1SclkL` (0): SCLK low period for Waveform 1.
- `kNi845xSpiStreamWave1SclkH` (1): SCLK high period for Waveform 1.
- `kNi845xSpiStreamWave1T1` (2): Timing Parameter T1—CONV assert to CONV deassert for Waveform 1.
- `kNi845xSpiStreamWave1T2` (3): Timing Parameter T2—CONV deassert to Chip Select assert for Waveform 1.
- `kNi845xSpiStreamWave1T3` (4): Timing Parameter T3—CONV deassert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T4` (5): Timing Parameter T4—DRDY assert to Chip Select assert for Waveform 1.
- `kNi845xSpiStreamWave1T5` (6): Timing Parameter T5—DRDY assert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T6` (7): Timing Parameter T6—DRDY deassert to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T7` (8): Timing Parameter T7—Chip Select assert to SCLK assert (first bit) for Waveform 1.
- `kNi845xSpiStreamWave1T8` (9): Timing Parameter T8—Chip Select deassert to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T9` (10): Timing Parameter T9—Chip Select deassert to Chip Select assert.

- `kNi845xSpiStreamWave1T10` (11): Timing Parameter T10—SCLK deassert (last bit) to CONV assert for Waveform 1.
- `kNi845xSpiStreamWave1T11` (12): Timing Parameter T11—SCLK deassert (last bit) to Chip Select deassert for Waveform 1.
- `kNi845xSpiStreamWave1T12` (13): Timing Parameter T12—SCLK deassert (last bit) to SCLK assert (first bit) for Waveform 1.

`uInt32 ParameterValue`

A 32-bit unsigned integer to set the timing parameter in system clocks.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamConfigurationWave1SetTimingParam` to set an individual timing parameter. Timing parameters are returned as number of system clocks. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for a description of the system clock and valid timing values on your NI 845*x* device.

SPI Stream API

ni845xSpiStreamRead

Purpose

Reads streaming data from an NI 845*x* device.

Format

```
int32 ni845xSpiStreamRead (
       uInt32    DeviceHandle,
       uInt32    ConfigurationHandle,
       uint32    NumBytesToRead,
       uInt8   * ReadData,
       uInt32  * ReadSize
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

`uInt32 NumBytesToRead`

The number of bytes to read. This number must be nonzero. `ReadData` must be large enough to read the requested number of bytes.

Outputs

`uInt8 * ReadData`

A pointer to an array of bytes where the bytes that have been read are stored.

`uInt32 * ReadSize`

A pointer to the amount of bytes actually read.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamRead` to read data from an SPI slave device. The read size is less than or equal to the value passed into `ReadSize` and is dependent on the packet size.

While your NI 845*x* device is in streaming mode, SPI operations continue to occur and buffer on board. `ni845xSpiStreamRead` does not affect SPI operations on the SPI. This function is reading the result of the streaming SPI operation started using `ni845xSpiStreamStart`.

ni845xSpiStreamStart

Purpose

Starts the streaming operation on an NI 845*x* device.

Format

```
int32 ni845xSpiStreamStart (
       uInt32 DeviceHandle,
       uInt32 ConfigurationHandle
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamStart` to put your NI 845*x* device into streaming mode. Once in streaming mode, your NI 845*x* device performs the SPI operations set to `ConfigurationHandle`. Your NI 845*x* device remains in streaming mode until `ni845xSpiStreamStop` is called.

The data set in `ni845xSpiStreamConfigurationWave1SetMosiData` is output on MOSI on each SPI operation during streaming. You can use this data to set up the SPI slave if necessary, but not all SPI slaves require it.

Before using `ni845xSpiStreamStart`, you must ensure that the configuration parameters specified in `ConfigurationHandle` are correct for the device you currently want to access.

ni845xSpiStreamStop

Purpose

Stops a streaming operation on an NI 845*x* device.

Format

```
int32 ni845xSpiStreamStop (
       uInt32  DeviceHandle,
       uInt32  ConfigurationHandle
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt32 ConfigurationHandle`

The configuration handle returned from `ni845xSpiStreamConfigurationOpen`.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSpiStreamStop` to remove your NI 845*x* device from streaming mode. When stopping, the device waits for the final SPI operation to complete if one is occurring. No data can be read from the device once stopped.

14

Using the NI-845*x* DIO API

This chapter helps you get started with the DIO API.

NI-845x DIO Basic Programming Model

When you use the DIO API, the first step is to configure the DIO port to be set for input or output as desired. Once the port is configured, you can write or read lines from the port. You can use either port or line I/O for all DIO calls. With the port calls, you can read or write all lines in a port at one time. Alternately, with the line calls, you can read or write the lines in a port one line at a time.

The diagram in Figure 14-1 describes the basic programming model for the NI-845x DIO API. Within the application, you repeat this basic programming model for each DIO call you need to make. The diagram is followed by a description of each step in the model.

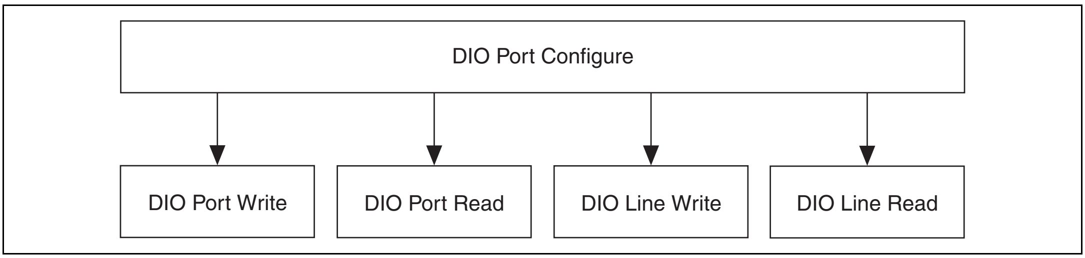


Figure 14-1. Basic Programming Model for DIO Communication

DIO Port Configure

The DIO Port configuration is set with the **NI-845x Device Property Node** in LabVIEW and `ni845xDioSet*` calls in other languages. The following parameters are available for configuring the DIO Port:

- **DIO:Active Port** (LabVIEW only) is the active DIO port to configure. The subsequent property settings affect only the selected DIO port.
- **DIO:Driver Type** configures the driver type used when sourcing DIO signals. The two options are open-drain and push-pull.

Note Not all NI 845*x* hardware supports all driver types.

- **DIO:Line Direction Map** indicates the direction (input or output) for each line in the 8-bit DIO port.
- **I/O Voltage Level** indicates the voltage level (when sourcing a high value) used for all push-pull I/O pins (SPI lines and DIO lines). It also affects the reference voltage that I^2C pins are pulled-up to if using internal I^2C pull-ups.

Note For other languages, this API call is `ni845xSetIoVoltageLevel` (this is a global property, not scoped to the DIO subsystem).

DIO Port Write

Use **NI-845x DIO Port Write.vi** in LabVIEW and `ni845xDioWritePort` in other languages to write an 8-bit pattern to the selected DIO port.

DIO Port Read

Use **NI-845x DIO Port Read.vi** in LabVIEW and `ni845xDioReadPort` in other languages to read an 8-bit pattern from the selected DIO port.

DIO Line Write

Use **NI-845x DIO Line Write.vi** in LabVIEW and `ni845xDioWriteLine` in other languages to write a value to a particular line within the selected DIO port.

DIO Line Read

Use **NI-845x DIO Line Read.vi** in LabVIEW and `ni845xDioReadLine` in other languages to read a value from a particular line within the selected DIO port.

15

NI-845*x* DIO API for LabVIEW

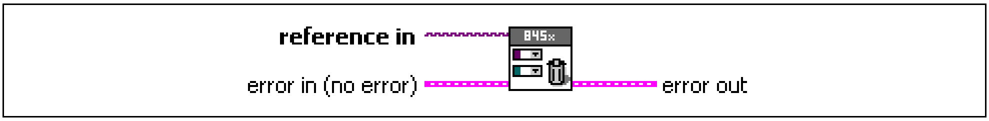
This chapter lists the LabVIEW VIs for the NI-845*x* DIO API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

General Device

NI-845x Close Reference.vi

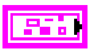
Purpose

Closes a previously opened reference.



Inputs

reference in is a reference to an NI 845*x* device, I²C configuration, SPI configuration, SPI stream configuration, I²C script, or SPI script.



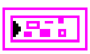
error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

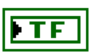
code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

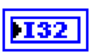
Outputs



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.



status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

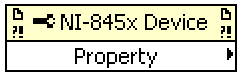
Description

Use **NI-845x Close Reference.vi** to close a previously opened reference.

NI-845x Device Property Node

Purpose

A property node with the NI-845*x* Device class preselected. This property node allows you to modify properties of your NI 845*x* device.



Inputs

device reference in is a reference to an NI 845*x* device.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to an NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is

returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

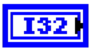
source identifies the VI where the error occurred.

Description

The list below describes all valid properties for the **NI-845x Device Property Node**.

DIO:Active Port

The **DIO:Active Port** property sets the active DIO port for further DIO port configuration. The format for this property is a decimal string. For example, the string `0` represents DIO Port 0. The default value of this property is `0`. For NI 845*x* devices with one DIO port, the port value must be `0`.



DIO:Driver Type

The **DIO:Driver Type** property configures the active DIO port with the desired driver type characteristics. **DIO:Driver Type** uses the following values:

`Open-Drain`

The DIO driver type is configured for open-drain.

`Push-Pull`

The DIO driver type is configured for push-pull. The actual voltage driven (when sourcing a high value) is determined by the I/O Voltage Level property.

The default value of this property is `Push-Pull`.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available driver types on your hardware.

DIO:Line Direction Map

The **DIO:Line Direction Map** property sets the line direction map for the active DIO Port. The value is a bitmap that specifies the function of each individual line within the port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

The default value of this property is `0` (all lines configured for input).

I/O Voltage Level

The **I/O Voltage Level** property sets the board voltage. This property sets the voltage for SPI, I^{2}C, and DIO. The default value for this property is `3.3V`. This property uses the following values:

`3.3V`

I/O Voltage is set to 3.3 V.

`2.5V`

I/O Voltage is set to 2.5 V.

`1.8V`

I/O Voltage is set to 1.8 V.

`1.5V`

I/O Voltage is set to 1.5 V.

`1.2V`

I/O Voltage is set to 1.2 V.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

I^{2}C Pullup Enable

The **I^{2}C Pullup Enable** property enables or disables the internal pullup resistors connected to SDA and SCL.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine whether your hardware has onboard pull-up resistors.

NI-845x Device Reference

Purpose

Specifies the device resource to be used for communication.

Description

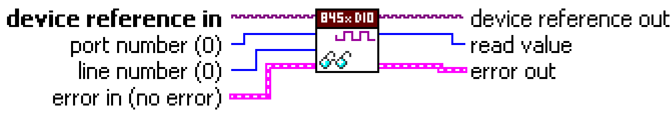
Use the NI-845*x* Device Reference to describe the NI 845*x* device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845*x* API call to invoke the function on the associated NI 845*x* device.

Basic

NI-845x DIO Read Line.vi

Purpose

Reads from a DIO line on an NI 845*x* device.

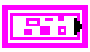


Inputs

device reference in is a reference to an NI 845*x* device.

port number specifies the DIO port that contains the **line number**.

line number specifies the DIO line to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

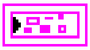
Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

read value is the value read from the line. **read value** uses the following values:

`0` (Logic Low) The line read is in the logic low state.

`1` (Logic High) The line read is in the logic high state.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

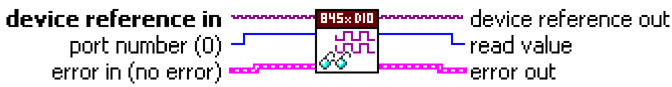
Description

Use **NI-845x DIO Read Line.vi** to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`). If **read value** is `0`, the logic level read on the specified line was low. If **read value** is `1`, the logic level read on the specified line was high.

NI-845x DIO Read Port.vi

Purpose

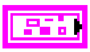
Reads from a DIO port on an NI 845*x* device.



Inputs

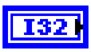
device reference in is a reference to an NI 845*x* device.

port number specifies the DIO port to read.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

read value is the value read from the DIO port. If a DIO pin was previously configured for input, the logic level being driven onto it by external circuitry is returned. If a DIO pin was previously configured for output, the logic level driven onto the pin internally is returned. **read value** bit n = DIO n.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

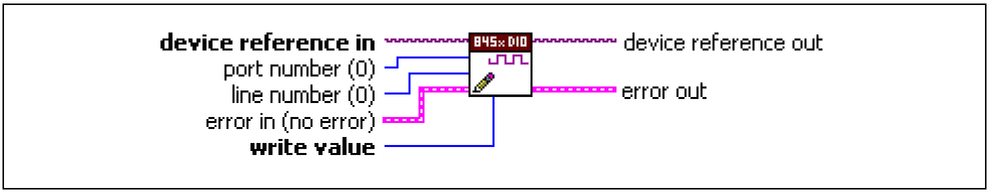
Description

Use **NI-845x DIO Read Port.vi** to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x DIO Write Line.vi

Purpose

Writes to a DIO line on an NI 845*x* device.

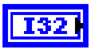


Inputs

device reference in is a reference to an NI 845*x* device.

port number specifies the DIO port that contains the **line number**.

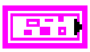
line number specifies the DIO line to write.



write value specifies the value to write to the line. **write value** uses the following values:

`0` (Logic Low) The line is set to the logic low state.

`1` (Logic High) The line is set to the logic high state.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

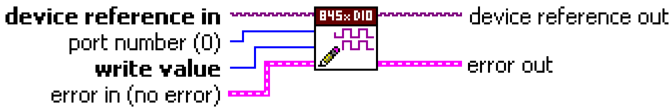
Description

Use **NI-845x DIO Write Line.vi** to write one line, specified by **line number**, of a byte-wide DIO port. If **write value** is `1`, the specified line's output is driven to a high logic level. If **write value** is `0`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

NI-845x DIO Write Port.vi

Purpose

Writes to a DIO port on an NI 845*x* device.

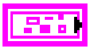


Inputs

device reference in is a reference to an NI 845*x* device.

port number specifies the DIO port to write.

write value is the value to write to the DIO port. Only lines configured for output are updated.



error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

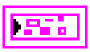
status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

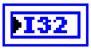
Outputs

device reference out is a reference to the NI 845*x* device after this VI runs.



error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.



code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use **NI-845x DIO Write Port.vi** to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845*x* devices with one DIO port, **port number** must be left at the default (`0`).

16

NI-845*x* DIO API for C

This chapter lists the functions for the NI-845*x* DIO API. The following topics describe the format, purpose, and parameters for each function. The functions are listed alphabetically in two categories: general device and basic.

Section Headings

The NI-845*x* DIO API for C functions include the following section headings.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Inputs and Outputs

These sections list the function input and output parameters.

Description

The description section gives details about the purpose and effect of each function.

Data Types

The NI-845*x* DIO API for C functions use the following data types.

Data Type	Purpose
`uInt8`	8-bit unsigned integer
`uInt16`	16-bit unsigned integer
`uInt32`	32-bit unsigned integer
`int8`	8-bit signed integer

Data Type	Purpose
`int16`	16-bit signed integer
`int32`	32-bit signed integer
`uInt8 *`	Pointer to an 8-bit unsigned integer
`uInt16 *`	Pointer to a 16-bit unsigned integer
`uInt32 *`	Pointer to a 32-bit unsigned integer
`int8 *`	Pointer to an 8-bit signed integer
`int16 *`	Pointer to a 16-bit signed integer
`int32 *`	Pointer to a 32-bit signed integer
`char *`	ASCII string represented as an array of characters terminated by null character (`'\0'`)

List of Functions

The following table contains an alphabetical list of the NI-845*x* DIO API for C functions.

Function	Purpose
`ni845xClose`	Closes a previously opened NI 845*x* device.
`ni845xCloseFindDeviceHandle`	Closes the handles created by `ni845xFindDevice`.
`ni845xDeviceLock`	Locks NI 845*x* devices for access by a single thread.
`ni845xDeviceUnlock`	Unlocks NI 845*x* devices.
`ni845xDioReadLine`	Reads from a DIO line on an NI 845*x* device.
`ni845xDioReadPort`	Reads from a DIO port on an NI 845*x* device.
`ni845xDioSetDriverType`	Configures the driver type used when sourcing DIO signals on an NI 845*x* device.
`ni845xDioSetPortLineDirectionMap`	Configures a DIO port on an NI 845*x* device for input or output.
`ni845xDioWriteLine`	Writes to a DIO line on an NI 845*x* device.
`ni845xDioWritePort`	Writes to a DIO port on an NI 845*x* device.

Function	Purpose
`ni845xFindDevice`	Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.
`ni845xFindDeviceNext`	Finds subsequent devices after `ni845xFindDevice` has been called.
`ni845xOpen`	Opens an NI 845*x* device for use with various write, read, and device property functions.
`ni845xStatusToString`	Converts a status code into a descriptive string.
`ni845xSetIoVoltageLevel`	Sets the voltage level of the NI-845*x* I/O pins (DIO/SPI/VioRef).

General Device

ni845xClose

Purpose

Closes a previously opened NI 845*x* device.

Format

```
int32 ni845xClose(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be closed.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xClose` to close a device handle previously opened by `ni845xOpen`. Passing an invalid handle to `ni845xClose` is ignored.

ni845xCloseFindDeviceHandle

Purpose

Closes the handles created by `ni845xFindDevice`.

Format

```
int32 ni845xCloseFindDeviceHandle (
      uInt32 FindDeviceHandle
      );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xCloseFindDeviceHandle` to close a find list. In this process, all allocated data structures are freed.

ni845xDeviceLock

Purpose

Locks NI 845*x* devices for access by a single thread.

Format

```
int32 ni845xDeviceLock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be locked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

This function locks NI 845*x* devices and prevents multiple processes or threads from accessing the device until the process or thread that owns the device lock calls an equal number of `ni845xDeviceUnlock` calls. Any thread or process that attempts to call `ni845xDeviceLock` when the device is already locked is forced to sleep by the operating system. This is useful for when multiple Basic API device accesses must occur uninterrupted by any other processes or threads. If a thread exits without fully unlocking the device, the device is unlocked. If a thread is the current owner of the lock, and calls `ni845xDeviceLock` again, the thread will not deadlock itself, but care must be taken to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` called. This function can possibly lock a device indefinitely: If a thread never calls `ni845xDeviceUnlock`, or fails to call `ni845xDeviceUnlock` for every `ni845xDeviceLock` call, and never exits, other processes and threads are forced to wait. This is *not* recommended for users unfamiliar with threads or processes. A simpler alternative is to use scripts. Scripts provide the same capability to ensure transfers are uninterrupted, and with possible performance benefits.

ni845xDeviceUnlock

Purpose

Unlocks NI 845*x* devices.

Format

```
int32 ni845xDeviceUnlock(uInt32 DeviceHandle);
```

Inputs

`uInt32 DeviceHandle`

Device handle to be unlocked.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDeviceUnlock` to unlock access to an NI 845*x* device previously locked with `ni845xDeviceLock`. Every call to `ni845xDeviceLock` must have a corresponding call to `ni845xDeviceUnlock`. Refer to `ni845xDeviceLock` for more details regarding how to use device locks.

ni845xFindDevice

Purpose

Finds an NI 845*x* device and returns the total number of NI 845*x* devices present. You can find subsequent devices using `ni845xFindDeviceNext`.

Format

```
int32 ni845xFindDevice (
      char   * pFirstDevice,
      uInt32 * pFindDeviceHandle,
      uInt32 * pNumberFound
      );
```

Inputs

None.

Outputs

`char * pFirstDevice`

A pointer to the string containing the first NI 845*x* device found. You can pass this name to the `ni845xOpen` function to open the device. If no devices exist, this is an empty string.

`uInt32 * pFindDeviceHandle`

Returns a handle identifying this search session. This handle is used as an input in `ni845xFindDeviceNext` and `ni845xCloseFindDeviceHandle`.

`uInt32 * pNumberFound`

A pointer to the total number of NI 845*x* devices found in the system. You can use this number in conjunction with the `ni845xFindDeviceNext` function to find a particular device. If no devices exist, this returns 0.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDevice` to get a single NI 845*x* device and the number of NI 845*x* devices in the system. You can then pass the string returned to `ni845xOpen` to access the device. If you must discover more devices, use `ni845xFindDeviceNext` with `pFindDeviceHandle` and `pNumberFound` to find the remaining NI 845*x* devices in the system. After finding all desired devices, call `ni845xCloseFindDeviceHandle` to close the device handle and relinquish allocated resources.

Note `pFirstDevice` must be at least 256 bytes.

Note `pFindDeviceHandle` and `pNumberFound` are optional parameters. If only the first match is important, and the total number of matches is not needed, you can pass in a NULL pointer for both of these parameters, and the NI-845x driver automatically calls `ni845xCloseFindDeviceHandle` before this function returns.

ni845xFindDeviceNext

Purpose

Finds subsequent devices after `ni845xFindDevice` has been called.

Format

```
int32 ni845xFindDeviceNext (
       uInt32 FindDeviceHandle,
       char * pNextDevice
       );
```

Inputs

`uInt32 FindDeviceHandle`

Describes a find list. `ni845xFindDevice` creates this parameter.

Outputs

`char * pNextDevice`

A pointer to the string containing the next NI 845*x* device found. This is empty if no further devices are left.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xFindDeviceNext` after first calling `ni845xFindDevice` to find the remaining devices in the system. You can then pass the string returned to `ni845xOpen` to access the device.

Note `pNextDevice` must be at least 256 bytes.

ni845xOpen

Purpose

Opens an NI 845*x* device for use with various write, read, and device property functions.

Format

```
int32 ni845xOpen (
        char   * pResourceName,
        uInt32 * pDeviceHandle
        );
```

Inputs

`char * pResourceName`

A resource name string corresponding to the NI 845*x* device to be opened.

Outputs

`uInt32 * pDeviceHandle`

A pointer to the device handle.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xOpen` to open an NI 845*x* device for access. The string passed to `ni845xOpen` can be any of the following: an `ni845xFindDevice` device string, an `ni845xFindDeviceNext` device string, a Measurement & Automation Explorer resource name, or a Measurement & Automation Explorer alias.

ni845xStatusToString

Purpose

Converts a status code into a descriptive string.

Format

```
void ni845xStatusToString (
       int32  StatusCode,
       uInt32 MaxSize,
       int8 * pStatusString
       );
```

Inputs

`int32 StatusCode`

Status code returned from an NI-845*x* function.

`uInt32 MaxSize`

Size of the `pStatusString` buffer (in bytes).

Outputs

`int8 * pStatusString`

ASCII string that describes `StatusCode`.

Description

When the status code returned from an NI-845*x* function is nonzero, an error or warning is indicated. This function obtains a description of the error/warning for debugging purposes.

The return code is passed into the `StatusCode` parameter. The `MaxSize` parameter indicates the number of bytes available in `pStatusString` for the description (including the NULL character). The description is truncated to size `MaxSize` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `String` is null-terminated, so you can use it with ANSI C functions such as `printf`.

For applications written in C or C++, each NI-845*x* function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845*x* use of this status.

NI-845*x* Status Codes

Status Code	Meaning
Negative	Error—Function did not perform expected behavior.
Positive	Warning—Function executed, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every NI-845*x* function. If an error is detected, you should close all NI-845*x* handles, then exit the application. If a warning is detected, you can display a message for debugging purposes, or simply ignore the warning.

In some situations, you may want to check for specific errors in the code and continue communication when they occur. For example, when communicating to an I^2C EEPROM, you may expect the device to NAK its address during a write cycle, and you may use this knowledge to poll for when the write cycle has completed.

Basic

ni845xDioReadLine

Purpose

Reads from a DIO line on an NI 845*x* device.

Format

```
int32 ni845xDioReadLine (
       uInt32   DeviceHandle,
       uInt8    PortNumber,
       uInt8    LineNumber,
       int32  * pReadData
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

`PortNumber` specifies the DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

`LineNumber` specifies the DIO line to read.

Outputs

`int32 * pReadData`

Contains the value read from the line. `pReadData` uses the following values:

- `kNi845xDioLogicLow` (0): The line is set to the logic low state.
- `kNi845xDioLogicHigh` (1): The line is set to the logic high state.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioReadLine` to read one line, specified by `LineNumber`, of a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0). If `pReadData` is `kNi845xDioLogicLow`, the logic level read on the specified line was low. If `pReadData` is `kNi845xDioLogicHigh`, the logic level read on the specified line was high.

ni845xDioReadPort

Purpose

Reads from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xDioReadPort (
       uInt32  DeviceHandle,
       uInt8   PortNumber,
       uInt8 * pReadData
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

`PortNumber` specifies the DIO port to read.

Outputs

`uInt8 * pReadData`

Contains the value read from the DIO port. If a DIO pin was previously configured for input, the logic level being driven onto it by external circuitry is returned. If a DIO pin was previously configured for output, the logic level driven onto the pin internally is returned. `pReadData` bit *n* = DIO *n*.

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioReadPort` to read all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xDioSetPortLineDirectionMap

Purpose

Configures a DIO port on an NI 845*x* device for input or output.

Format

```
int32 ni845xDioSetPortLineDirectionMap (
      uInt32 DeviceHandle,
      uInt8  DioPort,
      uInt8  Map
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 DioPort`

The DIO port that contains the `LineNumber`.

`uInt8 Map`

Sets the line direction map for the active DIO Port. The value is a bitmap that specifies the function of each individual line within the port. If bit $x = 1$, line x is an output. If bit $x = 0$, line x is an input.

The default value of this property is 0 (all lines configured for input).

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioSetPortLineDirectionMap` to modify a DIO port on an NI 845*x* device for input or output.

ni845xDioSetDriverType

Purpose

Configures the driver type used when sourcing DIO signals on an NI 845*x* device.

Format

```
int32 ni845xDioSetDriverType (
      uInt32 DeviceHandle,
      uInt8  DioPort,
      uInt8  Type
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 DioPort`

The DIO port that contains the `LineNumber`.

`uInt8 Type`

The desired output driver type. `Type` uses the following values:

- `kNi845xOpenDrain` (0): The port is configured for open-drain.
- `kNi845xPushPull` (1): The port is configured for push-pull.

The default value of this property is Push-Pull.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioSetDriverType` to modify the DIO driver type that the NI 845*x* devices use to source DIO signals. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which driver types your NI 845*x* device supports.

ni845xDioWriteLine

Purpose

Writes to a DIO line on an NI 845*x* device.

Format

```
int32 ni845xDioWriteLine (
      uInt32 DeviceHandle,
      uInt8  PortNumber,
      uInt8  LineNumber,
      int32  WriteData
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

The DIO port that contains the `LineNumber`.

`uInt8 LineNumber`

The DIO line to write.

`int32 WriteData`

The value to write to the line. `WriteData` uses the following values:

- `kNi845xDioLogicLow` (0): The line is set to the logic low state.
- `kNi845xDioLogicHigh` (1): The line is set to the logic high state.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioWriteLine` to write one line, specified by `LineNumber`, of a byte-wide DIO port. If `WriteData` is `kNi845xDioLogicHigh`, the specified line's output is driven to a high logic level. If `WriteData` is `kNi845xDioLogicLow`, the specified line's output is driven to a low logic level. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xDioWritePort

Purpose

Writes to a DIO port on an NI 845*x* device.

Format

```
int32 ni845xDioWritePort (
      uInt32 DeviceHandle,
      uInt8  PortNumber,
      uInt8  WriteData
      );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 PortNumber`

The DIO port to write.

`uInt8 WriteData`

The value to write to the DIO port. Only lines configured for output are updated.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xDioWritePort` to write all 8 bits on a byte-wide DIO port. For NI 845*x* devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845*x* devices with one DIO port, leave `PortNumber` at the default (0).

ni845xSetIoVoltageLevel

Purpose

Modifies the voltage output from a DIO port on an NI 845*x* device.

Format

```
int32 ni845xSetIoVoltageLevel (
       uInt32  DeviceHandle,
       uInt8   VoltageLevel
       );
```

Inputs

`uInt32 DeviceHandle`

Device handle returned from `ni845xOpen`.

`uInt8 VoltageLevel`

The desired voltage level. `VoltageLevel` uses the following values:

- `kNi845x33Volts` (33): The output I/O high level is 3.3 V.
- `kNi845x25Volts` (25): The output I/O high level is 2.5 V.
- `kNi845x18Volts` (18): The output I/O high level is 1.8 V.
- `kNi845x15Volts` (15): The output I/O high level is 1.5 V.
- `kNi845x12Volts` (12): The output I/O high level is 1.2 V.

The default value of this property is 3.3 V.

Outputs

Return Value

The function call status. Zero means the function executed successfully. Negative specifies an error, meaning the function did not perform the expected behavior. Positive specifies a warning, meaning the function performed as expected, but a condition arose that might require attention. For more information, refer to `ni845xStatusToString`.

Description

Use `ni845xSetIoVoltageLevel` to modify the board reference voltage of the NI 845*x* device. The board reference voltage is used for SPI, I^2C, and DIO. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine the available voltage levels on your hardware.

A

NI USB-845*x* Hardware Specifications

This appendix lists the NI USB-845*x* hardware specifications.

NI USB-8451

The following specifications are typical at 25 °C unless otherwise noted.

Digital I/O (DIO)

Number of lines

P0.<0..7> 8

Direction control .... Input or output, software selectable

Output driver type .... Push-pull (active drive) or open-drain, software selectable

Absolute voltage range........................... –0.5 to +5.8 V with respect to GND

Power-on state .... Input (high impedance)

Digital logic levels

Level	Min	Max	Units
Input			
Input low voltage	–0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	—	50	μA
Output			
Output low voltage (I = 8.5 mA)	—	0.8	V
Output high voltage			
Push-pull (active drive), I = –8.5 mA	2.0	3.5	V
Open-drain	V_{cc}[1]	V_{cc}[1]	V

[1] V_{cc} refers to the pull-up voltage you select.

SPI Interface

Signals

SPI CS <0..7>.................................Output

SPI MOSI (SDO).............................Output

SPI MISO (SDI) ....Input

SPI CLK (SCLK) ....Output (12 MHz max)

Supported clock rates..............................48 kHz, 50 kHz, 60 kHz, 75 kHz, 80 kHz, 96 kHz, 100 kHz, 120 kHz, 125 kHz, 150 kHz, 160 kHz, 200 kHz, 240 kHz, 250 kHz, 300 kHz, 375 kHz, 400 kHz, 480 kHz, 500 kHz, 600 kHz, 750 kHz, 800 kHz, 1 MHz, 1.2 MHz, 1.5 MHz, 2 MHz, 2.4 MHz, 3 MHz, 4 MHz, 6 MHz, 12 MHz

Output driver type...................................Push-pull (active drive)

Absolute voltage range ....–0.5 to +5.8 V with respect to GND

Power-on state ....Input (high impedance)

Digital logic levels

Level	Min	Max	Units
Input			
Input low voltage	–0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	—	50	μA
Output			
Output low voltage (I = 8.5 mA)	—	0.8	V
Output high voltage			
Push-pull (active drive), I = –8.5 mA	2.0	3.5	V

I^{2}C Interface

Signals

SDA .... Output/input

SCL .... Output (250 kHz max)

Supported clock rates

I^{2}C Standard Mode.......................... 32 kHz, 40 kHz, 50 kHz, 64 kHz, 80 kHz, 100 kHz

I^{2}C Fast Mode .... 125 kHz, 160 kHz, 200 kHz, 250 kHz

Fast Mode Plus............................... Not supported

I^{2}C High Speed Mode .... Not supported

Output driver type .... Open-drain

Absolute voltage range.......................... –0.5 to +5.8 V with respect to GND

Power-on state....................................... Input (high impedance)

Digital logic levels

Level	Min	Max	Units
Output			
Output low voltage (I = 8.5 mA)	—	0.8	V
Output high voltage			
Open-drain with external pull-up resistor	2.0	—	V

Note This interface is compatible with both I^{2}C and SMBus devices.

Bus Interface

USB specification ....Full-speed (12 Mb/s)

Power Requirements

USB

4.10 to 5.25 VDC............................80 mA typical, 500 mA max

USB Suspend..................................300 µA standby mode, 500 µA max

Output Voltage Sources

+5 V output

Voltage ....4.10 V min, 5.25 V max

Current...230 mA max

Physical Characteristics

NI USB-8451

Dimensions

Without connectors..........................6.35 cm × 8.51 cm × 2.31 cm (2.50 in. × 3.35 in. × 0.91 in.)

With connectors..............................8.18 cm × 8.51 cm × 2.31 cm (3.22 in. × 3.35 in. × 0.91 in.)

I/O connectors..USB series B receptacle, two 16-position (screw terminal) plug headers

Screw-terminal wiring ....16 AWG to 28 AWG copper conductor wire with 10 mm (0.39 in.) of insulation stripped from the end

Torque for screw terminals.....................0.22 to 0.25 N • m (2.0 to 2.2 lb • in.)

Weight ....84 g (3 oz)

NI USB-8451 OEM

Dimensions.. 5.74 cm × 6.73 cm × 1.15 cm
(2.26 in. × 2.65 in. × 0.45 in.)

I/O connectors .... USB series B receptacle;
34-p IDC ribbon cable header

Weight.. 21 g (0.74 oz)

Dimensional drawings

Figure A-1 shows a top view of the USB-8451 OEM. Figure A-2 shows the front and rear dimensions.

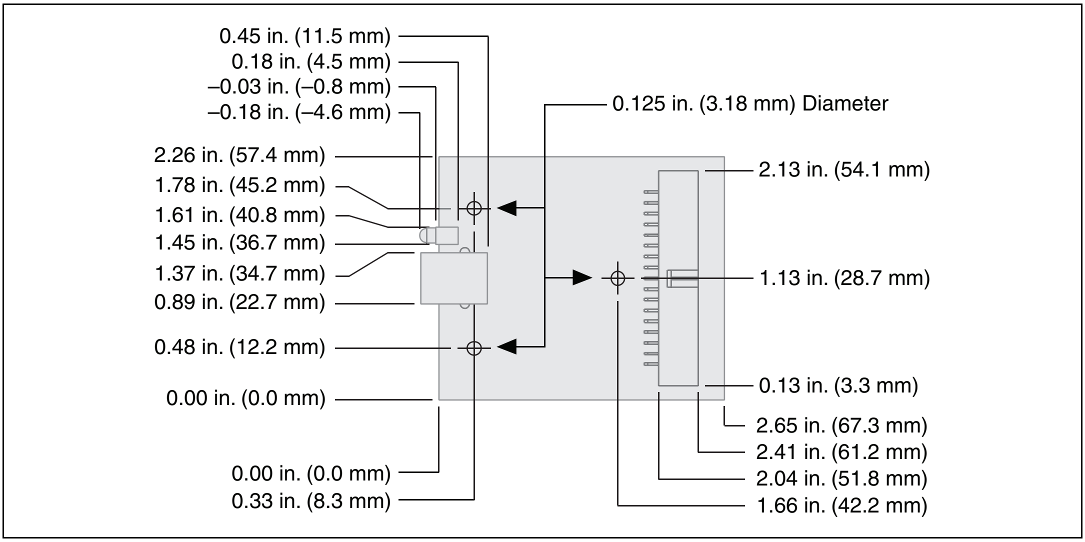


Figure A-1. USB-8451 OEM Dimensions (Top View)

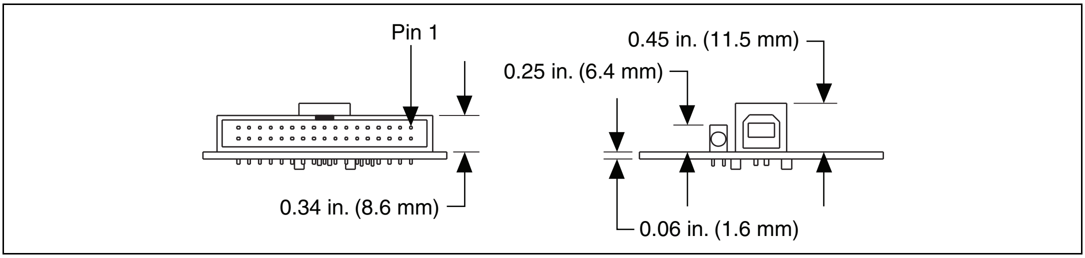


Figure A-2. USB-8451 OEM Dimensions (Front and Rear Views)

Overvoltage Protection

Connect only voltages that are within these limits.

Channel-to-COM (one channel) ....± 30 V max,
Measurement Category I

Channels-to-COM
(one port, all channels) .... ± 8.9 V max,
Measurement Category I

Measurement Category I is for measurements performed on circuits not directly connected to the electrical distribution system referred to as MAINS voltage. MAINS is a hazardous live electrical supply system that powers equipment. This category is for measurements of voltages from specially protected secondary circuits. Such voltage measurements include signal levels, special equipment, limited-energy parts of equipment, circuits powered by regulated low-voltage sources, and electronics.

Caution Do not use this module for connection to signals or for measurements within Measurement Categories II, III, or IV.

NI USB-8452

The following specifications are typical at 25 °C, unless otherwise noted.

Digital I/O(DIO)

Number of lines
DIO <0..7> 8

Direction control....................................Input or output, software selectable

Output driver type..................................Push-pull (active drive)

Absolute voltage range ....–0.5 to +5.5 V with respect to GND

Power-on state ....Tri-state with weak (40 kΩ) pull down to GND

I/O specifications under different logic levels

Output Specifications			
Logic Family	Voltage Low Level (V_{OL}) (Full Temperature) Max (I_{OL} = 100 uA)	Voltage High Level (V_{OH}) (Full Temperature) Min (I_{OH} = 100 uA)	Output Drive Strength (I_{O_MAX}) Max
1.2 V	0.2 V	1.0 V	±3 mA
1.5 V	0.2 V	1.3 V	±6 mA
1.8 V	0.2 V	1.6 V	±8 mA
2.5 V	0.2 V	2.3 V	±9 mA
3.3 V	0.2 V	3.1 V	±12 mA
Output Impedance	70 Ω (typical)		

Input Specifications		
Logic Family	Input Voltage Low (V_{IL}) Max	Input Voltage High (V_{IH}) Min
1.2 V	0.42 V	0.78 V
1.5 V	0.525 V	0.975 V
1.8 V	0.63 V	1.17 V
2.5 V	0.7 V	1.6 V
3.3 V	0.8 V	2 V
Input Impedance	High impedance	
Input Protection	–0.5 V to +5.5 V, ±50 mA maximum	

SPI Interface

Signals

SPI CS <0..7> .... Output

SPI MOSI (SDO) .... Output

SPI MISO (SDI).............................. Input

SPI CLK (SCLK)............................ Output (50 MHz max)

Supported clock rates	25 kHz, 32 kHz, 40 kHz, 50 kHz, 80 kHz, 100 kHz, 125 kHz, 160 kHz, 200 kHz, 250 kHz, 400 kHz, 500 kHz, 625 kHz, 800 kHz, 1 MHz, 1.25 MHz, 2.5 MHz, 3.125 MHz, 4 MHz, 5 MHz, 6.25 MHz, 10 MHz, 12.5 MHz, 20 MHz, 25 MHz, 33.33 MHz, 50 MHz
Output driver type	Push-pull (active drive)
Absolute voltage range	–0.5 to +5.5 V with respect to GND
Power-on state	Tri-state with weak (40 kΩ) pull down to GND
Transfer size	4 to 64 bits programmable (API specific)
Bit ordering	Most significant bit (msb) first

SPI specifications under different logic levels

Output Specifications			
	Voltage Low Level (V_{OL}) (Full Temperature)	**Voltage High Level (V_{OH}) (Full Temperature)**	**Output Drive Strength (I_{O_MAX})**
Logic Family	**Max (I_{OL} = 100 uA)**	**Min (I_{OH} = 100 uA)**	**Max**
1.2 V	0.2 V	1.0 V	±3 mA
1.5 V	0.2 V	1.3 V	±6 mA
1.8 V	0.2 V	1.6 V	±8 mA
2.5 V	0.2 V	2.3 V	±9 mA
3.3 V	0.2 V	3.1 V	±12 mA
Output Impedance	70 Ω (typical)		

Input Specifications		
Logic Family	**Input Voltage Low (V_{IL}) Max**	**Input Voltage High (V_{IH}) Min**
1.2 V	0.42 V	0.78 V
1.5 V	0.525 V	0.975 V
1.8 V	0.63 V	1.17 V
2.5 V	0.7 V	1.6 V
3.3 V	0.8 V	2 V
Input Impedance	High impedance	
Input Protection	–0.5 V to +5.5 V, ±50 mA maximum	

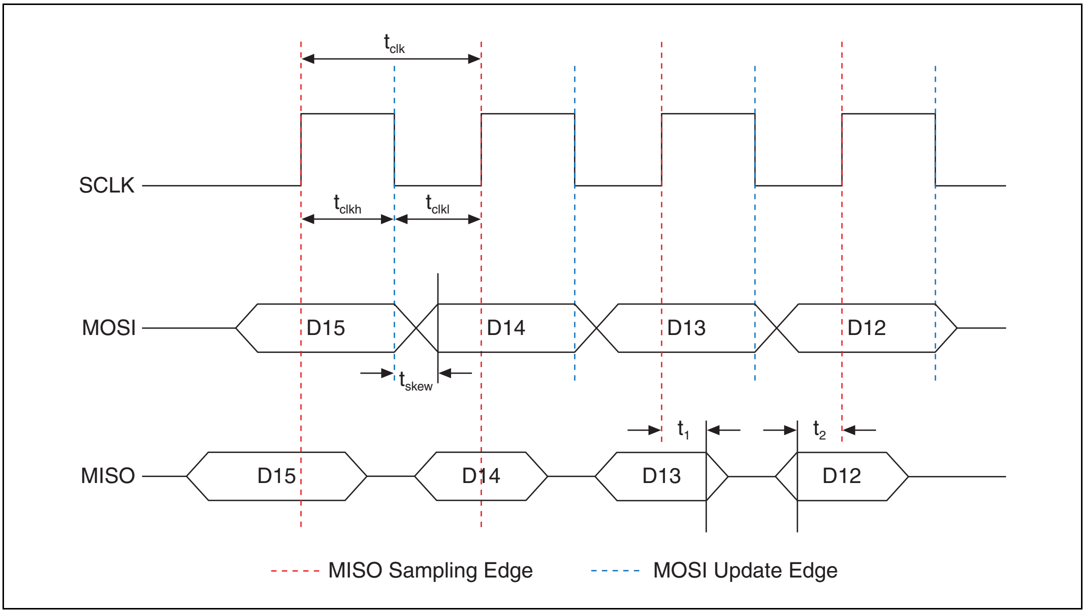
SPI timing requirements

Timing Parameter[1]	Min	Max	Unit
t_{clk} SCLK period	20	—	ns
t_{clkl} SCLK low time	9	—	ns
t_{clkh} SCLK high time	9	—	ns
t_{skew} MOSI output skew (with regard to SCLK edge)	–2	2	ns

Timing Parameter[1]	Min	Max	Unit
t_1 MISO hold time	5	—	ns
t_2 MISO setup time	4	—	ns

[1] All timing parameters are measured/required at IDC connector.

SPI timing diagram



I²C Interface

Signals

SDA ....Output/input

SCL..Output/input (3.3 MHz max)

Supported clock rates

I²C Standard Mode ....16 kHz, 20 kHz, 25 kHz, 31 kHz, 40 kHz, 50 kHz, 62 kHz, 80 kHz, 100 kHz

I²C Fast Mode ....125 kHz, 200 kHz, 250 kHz, 400 kHz

I^2C Fast Mode Plus .... 500 kHz, 1 Mhz

I^2C High Speed Mode .... 1.11 MHz, 1.33 MHz, 2.22 MHz, 3.33 MHz

Output driver type .... Open-drain

Absolute voltage range........................... –0.5 V to +5.5 V with respect to GND

Absolute input current............................ 40 mA max

Power-on state....................................... High impedance without pull-up

I^2C specifications under different logic levels

Logic Family	**Output Voltage Low (V_{OL}) Max**	**Input Voltage Low (V_{IL}) Max**
1.2 V	0.2 V	0.4 V
1.5 V	0.2 V	0.4 V
1.8 V	0.2 V	0.4 V
2.5 V	0.2 V	0.4 V
3.3 V	0.2 V	0.4 V
Pull-up current	3 mA (max)[1]	
Onboard capacitance	70 pF (max)	
Input protection	40 mA (max)	
[1] With onboard pull-up resistors enabled (tested under V_{OL} = 0.24 V)		

Note This interface is compatible with both I^2C and SMBus devices. (SMBus compatibility is only under Vref= 3.3 V and using external pull-up resistors instead of onboard pull-ups. For a proper pull-up value, refer to the SMBus specifications.)

Bus Interface

USB specification .... USB 2.0 High-Speed (480 Mb/s)

Power Requirements

USB high-power bus-powered device

Input voltage....................................4.5 V min, 5.25 V max

Working mode current.....................500 mA maximum,
250 mA typical

USB suspend ....2.5 mA maximum (all front
I/O lines disconnected)

Output Voltage Sources

+5 V output

Voltage ....4.75 V min, 5.25 V max

Current..20 mA max

Vref I/O reference output

Voltage ....1.2 V, 1.5 V, 1.8 V, 2.5 V,
3.3 V programmable, with
±10% tolerance

Current..20 mA max

Physical Characteristics

Dimensions

Without connector(s).......................3.39 in. × 2.62 in.
(8.61 cm × 6.65 cm)

With connector(s)............................3.49 in. × 2.62 in.
(8.86 cm × 6.65 cm)

I/O connectors ....1 × right angle USB series B
receptacle
1 × right angle male IDE cable
receptacle

Weight ....35 g (1.23 oz)

Dimensional drawings

Figure A-3 shows a top view of the NI USB-8452 OEM. Figure A-4 shows the front and rear dimensions.

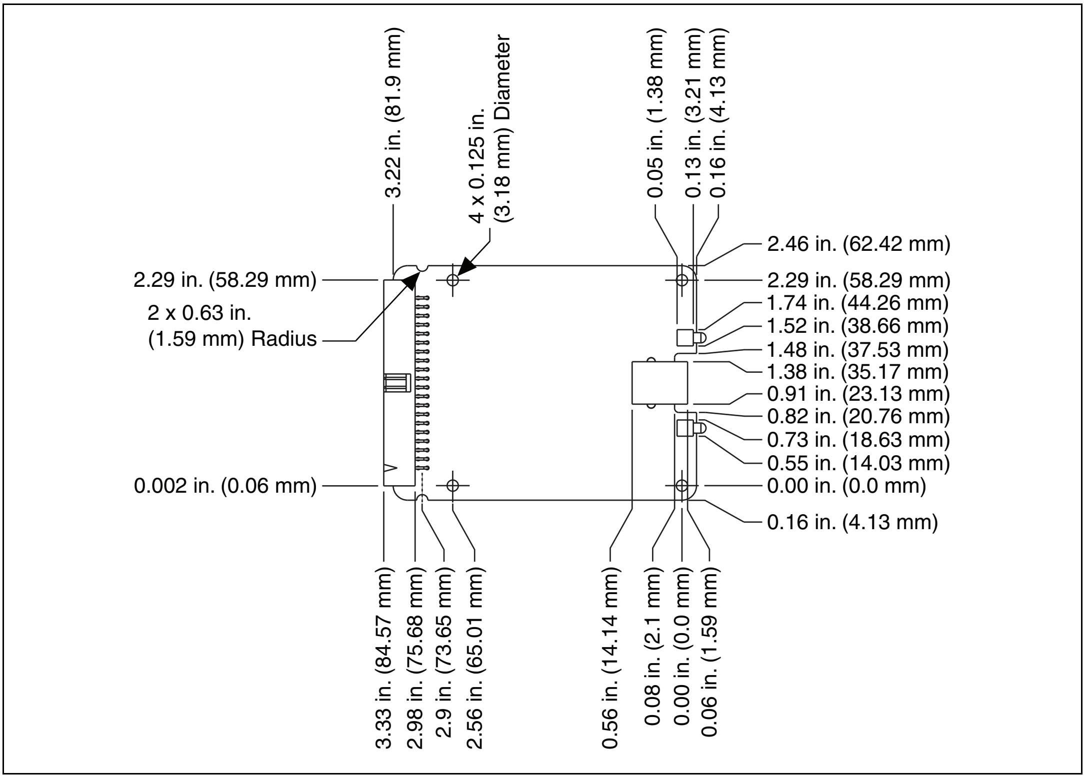


Figure A-3. USB-8452 OEM Dimensions (Top View)

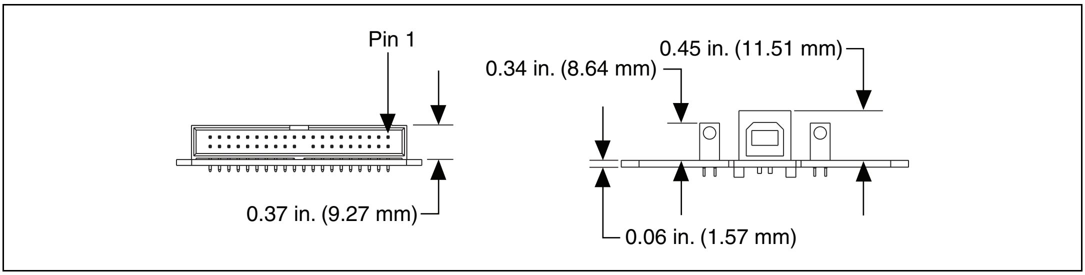


Figure A-4. USB-8452 OEM Dimensions (Front and Rear Views)

Safety

Safety Standards

This product meets the requirements of the following standards of safety for electrical equipment for measurement, control, and laboratory use:

- IEC 61010-1, EN 61010-1
- UL 61010-1, CSA 61010-1

Note For UL and other safety certifications, refer to the product label or the *Online Product Certification* section.

Hazardous Locations

The NI USB-845*x* modules are not certified for use in hazardous locations.

Electromagnetic Compatibility

NI USB-8451

This product meets the requirements of the following EMC standards for electrical equipment for measurement, control, and laboratory use:

- EN 61326 (IEC 61326): Class A emissions; Basic immunity
- EN 55011 (CISPR 11): Group 1, Class A emissions
- AS/NZS CISPR 11: Group 1, Class A emissions
- FCC 47 CFR Part 15B: Class A emissions
- ICES-001: Class A emissions

Note For the standards applied to assess the EMC of this product, refer to the *Online Product Certification* section.

Note For EMC compliance, operate this product according to the documentation.

NI USB-8451 OEM, NI USB-8452 OEM

The NI USB-8451 OEM and NI USB-8452 OEM devices are intended for use as part of a system. To ensure that your system meets the appropriate EMC standards, you must test the entire system.

CE Compliance CE

NI USB-8451

This product meets the essential requirements of applicable European Directives as follows:

- 2006/95/EC; Low-Voltage Directive (safety)
- 2004/108/EC; Electromagnetic Compatibility Directive (EMC)

NI USB-8451 OEM, NI USB-8452 OEM

The NI USB-8451 OEM and NI USB-8452 OEM devices are intended for use as part of a system. To ensure that your system meets the appropriate CE Compliance regulations, you must test the entire system.

Online Product Certification

Refer to the product Declaration of Conformity (DoC) for additional regulatory compliance information. To obtain product certifications and the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the Certification column.

Environmental

The NI USB-845*x* modules are intended for indoor use only.

Operating temperature
(IEC 60068-2-1 and IEC 60068-2-2) ..... 0 to 45 °C

Operating humidity (IEC 60068-2-56) .. 10 to 90% RH, noncondensing

Maximum altitude .... 2,000 m (at 25°C ambient temperature)

Storage temperature (IEC 60068-2-1
and IEC 60068-2-2) .... –40 to 85 °C

Storage humidity (IEC 60068-2-56) ...... 5 to 90% RH, noncondensing

Pollution Degree (IEC 60664) 2

Environmental Management

NI is committed to designing and manufacturing products in an environmentally responsible manner. NI recognizes that eliminating certain hazardous substances from our products is beneficial to the environment and to NI customers.

For additional environmental information, refer to the *NI and the Environment* Web page at `ni.com/environment`. This page contains the environmental regulations and directives with which NI complies, as well as other environmental information not included in this document.

Waste Electrical and Electronic Equipment (WEEE)

EU Customers At the end of the product life cycle, all products *must* be sent to a WEEE recycling center. For more information about WEEE recycling centers, National Instruments WEEE initiatives, and compliance with WEEE Directive 2002/96/EC on Waste and Electronic Equipment, visit `ni.com/environment/weee`.

电子信息产品污染控制管理办法 （中国 RoHS）

中国客户 National Instruments 符合中国电子信息产品中限制使用某些有害物质指令 (RoHS)。关于 National Instruments 中国 RoHS 合规性信息，请登录 `ni.com/environment/rohs_china`。(For information about China RoHS compliance, go to `ni.com/environment/rohs_china`.)

B

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Technical support at `ni.com/support` includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit `ni.com/support` for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at `ni.com/forums`. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support as well as exclusive access to on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

    For information about other technical support options in your area, visit `ni.com/services`, or contact your local office at `ni.com/contact`.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

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

A

Arbitration — The procedure to allow multiple masters to determine which single master controls the bus for a particular transfer time.

C

CLK — CLocK. The clock is generated by the master device and controls when data is sent and read.

CPHA — Clock PHAse. This controls the positioning of the data bits relative to the clock edges.

CPOL — Clock POLarity. The polarity indicating whether the clock makes positive or negative pulses.

CS or SS — Chip Select or Slave Select. Connection from the master to a slave that signals the slave to listen for SPI clock and data signals.

I

I^2C	Inter-IC

M

Master	On the I^2C bus, a device that can initiate and terminate a transfer on the bus. The master is responsible for generating the clock (SCL) signal. On the SPI bus, the master device provides the clock signal and determines the chip select line state.
MISO	Master Input, Slave Output. The MISO carries data from the slave to the master.
MOSI	Master Output, Slave Input. The MOSI line carries data from the master to the slave.
Multimaster	The ability for more than one master to co-exist on the bus concurrently without data loss.

R

Receiver	Device receiving data from the bus.

S

SCL	Serial CLock (clock signal line).
SDA	Serial DAta (data signal line).
Shift Register	A shift register is connected to the MOSI and MISO lines. As data is read from the input, it is placed into the shift register. Data from the shift register is placed into the output, creating a full-duplex communication loop.
Slave	On the I^2C bus, a device addressed by the master. On the SPI bus, the slave device receives the clock and chip select from the master. The maximum number of slaves is dependent on the number of available chip select lines.

SMBus	System Management Bus
Synchronization	The defined procedure to allow the clock signals provided by two or more masters to be synchronized.

T

Transmitter	Device transmitting data on the bus.

Index

D

E

N

T

V

W