

## COMPREHENSIVE SERVICES

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

## SELL YOUR SURPLUS

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

 Sell For Cash  Get Credit  Receive a Trade-In Deal

## OBSOLETE NI HARDWARE IN STOCK & READY TO SHIP

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



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

 1-800-915-6216

 [www.apexwaves.com](http://www.apexwaves.com)

 [sales@apexwaves.com](mailto:sales@apexwaves.com)

*All trademarks, brands, and brand names are the property of their respective owners.*

**Request a Quote**

 **CLICK HERE**

**USB-8451**

# I<sup>2</sup>C/SPI

## NI-845x Hardware and Software Manual

## **Worldwide Technical Support and Product Information**

[ni.com](http://ni.com)

## **Worldwide Offices**

Visit [ni.com/niglobal](http://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](http://ni.com/info) and enter the Info Code `feedback`.

# Important Information

---

## Warranty

The USB-845x 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 THEREFORE 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

## End-User License Agreements and Third-Party Legal Notices

You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:

- Notices are located in the <National Instruments>\\_Legal Information and <National Instruments> directories.
- EULAs are located in the <National Instruments>\Shared\MDF\Legal\license directory.
- Review <National Instruments>\\_Legal Information.txt for more information on including legal information in installers built with NI products.

## Trademarks

LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the *Trademark Information* at [ni.com/trademarks](http://ni.com/trademarks) for other National Instruments trademarks.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

## 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](http://ni.com/patents).

## Export Compliance Information

Refer to the *Export Compliance Information* at [ni.com/legal/export-compliance](http://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)<sup>1</sup>. 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<sup>1</sup>.

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<sup>1</sup> 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.

---

<sup>1</sup> 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](http://ni.com/certification), search by model number or product line, and click the appropriate link in the Certification column.

# Contents

---

## About This Manual

Conventions .....	xxi
-------------------	-----

## Chapter 1 Introduction

I <sup>2</sup> C Bus .....	1-1
I <sup>2</sup> C Terminology .....	1-1
I <sup>2</sup> C Bus .....	1-2
I <sup>2</sup> C Arbitration .....	1-2
I <sup>2</sup> C Transfers .....	1-3
I <sup>2</sup> C ACK Polling .....	1-4
I <sup>2</sup> C Clock Stretching .....	1-4
I <sup>2</sup> C Extended (10-Bit) Addressing .....	1-4
I <sup>2</sup> C High Speed Master Code .....	1-5
I <sup>2</sup> C vs. SMBus .....	1-5
SPI Bus .....	1-6
SPI Terminology .....	1-6
SPI Bus .....	1-7
Clock and Polarity .....	1-7
Error Handling .....	1-8

## Chapter 2 Installation

Software Installation .....	2-1
Hardware Installation .....	2-1
System Configuration API .....	2-3

## Chapter 3 NI USB-845x Hardware Overview

Overview .....	3-1
NI USB-8451 .....	3-1
Overview .....	3-1
Block Diagram .....	3-2
Installing Software .....	3-2
Setting Up Hardware .....	3-2
NI USB-8451 .....	3-2
NI USB-8451 OEM .....	3-3

I/O Connector and Cable.....	3-4
NI USB-8451 .....	3-4
NI USB-8451 OEM .....	3-5
Signal Descriptions .....	3-6
Front-End I/O Interfaces .....	3-7
Digital I/O (DIO) .....	3-7
SPI Interface .....	3-9
I <sup>2</sup> C Interface.....	3-10
I/O Protection.....	3-10
Power-On States.....	3-11
+5 V Power Source .....	3-11
NI USB-8452.....	3-11
Overview .....	3-11
Block Diagram .....	3-12
Installing Software .....	3-13
Setting Up Hardware.....	3-13
Signal Descriptions .....	3-14
Front-End I/O Interfaces .....	3-16
SPI Interface .....	3-16
I <sup>2</sup> C Interface.....	3-18
Digital I/O (DIO) .....	3-19
LED Indicators .....	3-20
I/O Protection.....	3-21
Power-On States.....	3-21
Power Sources.....	3-21
+5 V Power Source.....	3-21
Vref I/O Reference Voltage.....	3-22

## Chapter 4 Using the NI-845x API

## Chapter 5 Using the NI-845x I<sup>2</sup>C API

I <sup>2</sup> C Basic Programming Model .....	5-1
I <sup>2</sup> C Configure .....	5-2
I <sup>2</sup> C Write .....	5-2
I <sup>2</sup> C Read .....	5-2
I <sup>2</sup> C Write Read .....	5-2
I <sup>2</sup> C Scripting Programming Model.....	5-2
Script: Set I <sup>2</sup> C Clock Rate.....	5-4
Script: Set I <sup>2</sup> C ACK Poll Timeout .....	5-4
Script: Pullup Enable .....	5-4



Script: Set I2C High Speed Clock Rate.....	5-4
Script: Set I2C High Speed Enable .....	5-4
Script: Issue Start Condition.....	5-5
Script: Send High Speed Master Code .....	5-5
Script: Send Address + Read.....	5-5
Script: Read .....	5-5
Script: Send Address + Write.....	5-5
Script: Write .....	5-6
Script: Issue Stop Condition.....	5-6
Run Script.....	5-6
Extract Read Data.....	5-6

## Chapter 6

### NI-845x I<sup>2</sup>C API for LabVIEW

General Device .....	6-2
NI-845x Close Reference.vi .....	6-2
NI-845x Device Property Node.....	6-4
NI-845x Device Reference .....	6-7
Configuration .....	6-8
NI-845x I2C Configuration Property Node.....	6-8
NI-845x I2C Create Configuration Reference.vi .....	6-12
Basic.....	6-14
NI-845x I2C Read.vi .....	6-14
NI-845x I2C Write.vi .....	6-16
NI-845x I2C Write Read.vi .....	6-18
Scripting.....	6-20
NI-845x I2C Create Script Reference.vi .....	6-20
NI-845x I2C Extract Script Read Data.vi.....	6-22
NI-845x I2C Run Script.vi .....	6-24
NI-845x I2C Script ACK Poll Timeout.vi .....	6-26
NI-845x I2C Script Address+Read.vi .....	6-28
NI-845x I2C Script Address+Write.vi .....	6-30
NI-845x I2C Script Clock Rate.vi .....	6-32
NI-845x I2C Script Delay (Microsecond).vi.....	6-34
NI-845x I2C Script Delay (Millisecond).vi.....	6-36
NI-845x I2C Script DIO Configure Line.vi .....	6-38
NI-845x I2C Script DIO Configure Port.vi.....	6-40
NI-845x I2C Script DIO Read Line.vi .....	6-42
NI-845x I2C Script DIO Read Port.vi.....	6-44
NI-845x I2C Script DIO Write Line.vi .....	6-46
NI-845x I2C Script DIO Write Port.vi.....	6-48
NI-845x I2C Script Pullup Enable.vi .....	6-50
NI-845x I2C Script HS Enable.vi.....	6-52

NI-845x I2C Script HS Master Code.vi.....	6-54
NI-845x I2C Script HS Clock Rate.vi .....	6-56
NI-845x I2C Script Issue Start.vi.....	6-58
NI-845x I2C Script Issue Stop.vi.....	6-60
NI-845x I2C Script Read.vi .....	6-62
NI-845x I2C Script Write.vi .....	6-64

## Chapter 7

### NI-845x I<sup>2</sup>C API for C

Section Headings .....	7-1
Purpose .....	7-1
Format .....	7-1
Inputs and Outputs .....	7-1
Description .....	7-1
Data Types.....	7-1
List of Functions.....	7-2
General Device .....	7-8
ni845xClose .....	7-8
ni845xCloseFindDeviceHandle .....	7-9
ni845xDeviceLock .....	7-10
ni845xDeviceUnlock .....	7-11
ni845xFindDevice.....	7-12
ni845xFindDeviceNext .....	7-14
ni845xOpen .....	7-15
ni845xSetIoVoltageLevel .....	7-16
ni845xI2cSetPullupEnable.....	7-17
ni845xStatusToString.....	7-18
ni845xSetTimeout.....	7-20
Configuration.....	7-21
ni845xI2cConfigurationClose.....	7-21
ni845xI2cConfigurationGetAckPollTimeout.....	7-22
ni845xI2cConfigurationGetAddress .....	7-23
ni845xI2cConfigurationGetAddressSize .....	7-24
ni845xI2cConfigurationGetClockRate .....	7-25
ni845xI2cConfigurationGetHSClockRate .....	7-26
ni845xI2cConfigurationGetHSEnable .....	7-27
ni845xI2cConfigurationGetHSMasterCode.....	7-28
ni845xI2cConfigurationGetPort.....	7-29
ni845xI2cConfigurationOpen .....	7-30
ni845xI2cConfigurationSetAckPollTimeout .....	7-31
ni845xI2cConfigurationSetAddress .....	7-32
ni845xI2cConfigurationSetAddressSize .....	7-33
ni845xI2cConfigurationSetClockRate .....	7-34

ni845xI2cConfigurationSetHSClockRate .....	7-35
ni845xI2cConfigurationSetHSEnable .....	7-36
ni845xI2cConfigurationSetHSMasterCode.....	7-37
ni845xI2cConfigurationSetPort.....	7-38
Basic.....	7-39
ni845xI2cRead.....	7-39
ni845xI2cWrite.....	7-41
ni845xI2cWriteRead.....	7-43
Scripting.....	7-45
ni845xI2cScriptAckPollTimeout.....	7-45
ni845xI2cScriptAddressRead .....	7-46
ni845xI2cScriptAddressWrite .....	7-47
ni845xI2cScriptClockRate .....	7-48
ni845xI2cScriptClose .....	7-49
ni845xI2cScriptDelay.....	7-50
ni845xI2cScriptDioConfigureLine.....	7-51
ni845xI2cScriptDioConfigurePort .....	7-52
ni845xI2cScriptDioReadLine.....	7-53
ni845xI2cScriptDioReadPort .....	7-55
ni845xI2cScriptDioWriteLine.....	7-56
ni845xI2cScriptDioWritePort.....	7-58
ni845xI2cScriptPullupEnable.....	7-59
ni845xI2cScriptExtractReadData .....	7-60
ni845xI2cScriptExtractReadDataSize .....	7-61
ni845xI2cScriptHSEnable .....	7-62
ni845xI2cScriptHSMasterCode.....	7-63
ni845xI2cScriptHSClockRate .....	7-64
ni845xI2cScriptIssueStart.....	7-65
ni845xI2cScriptIssueStop.....	7-66
ni845xI2cScriptOpen.....	7-67
ni845xI2cScriptRead .....	7-68
ni845xI2cScriptReset .....	7-70
ni845xI2cScriptRun.....	7-71
ni845xI2cScriptUsDelay .....	7-72
ni845xI2cScriptWrite .....	7-73

## Chapter 8

### Using the NI-845x I<sup>2</sup>C Slave API

I <sup>2</sup> C Slave Programming Model.....	8-1
I <sup>2</sup> C Slave Configure.....	8-2
I <sup>2</sup> C Slave Start .....	8-2
I <sup>2</sup> C Slave Wait For Event .....	8-3
I <sup>2</sup> C Slave Read.....	8-3

I <sup>2</sup> C Slave Write .....	8-3
I <sup>2</sup> C Slave Get Write Information .....	8-3
I <sup>2</sup> C Slave Stop .....	8-3

## Chapter 9

### NI-845x I<sup>2</sup>C Slave API for LabVIEW

General Device .....	9-2
NI-845x Close Reference.vi .....	9-2
NI-845x Device Property Node .....	9-4
NI-845x Device Reference .....	9-7
Configuration .....	9-8
NI-845x I2C Slave Configuration Property Node .....	9-8
NI-845x I2C Slave Create Configuration Reference.vi .....	9-11
NI-845x I2C Slave Get Write Info.vi .....	9-13
NI-845x I2C Slave Read.vi .....	9-15
NI-845x I2C Slave Start.vi .....	9-17
NI-845x I2C Slave Stop.vi .....	9-19
NI-845x I2C Slave Wait for Event.vi .....	9-21
NI-845x I2C Slave Write.vi .....	9-24

## Chapter 10

### NI-845x I<sup>2</sup>C Slave API for C

Section Headings .....	10-1
Purpose .....	10-1
Format .....	10-1
Inputs and Outputs .....	10-1
Description .....	10-1
Data Types .....	10-1
List of Functions .....	10-2
General Device .....	10-5
ni845xClose .....	10-5
ni845xCloseFindDeviceHandle .....	10-6
ni845xDeviceLock .....	10-7
ni845xDeviceUnlock .....	10-8
ni845xFindDevice .....	10-9
ni845xFindDeviceNext .....	10-11
ni845xOpen .....	10-12
ni845xSetIoVoltageLevel .....	10-13
ni845xI2cSetPullupEnable .....	10-14
ni845xSetTimeout .....	10-15
ni845xStatusToString .....	10-16

Configuration .....	10-18
ni845xI2cSlaveConfigurationClose .....	10-18
ni845xI2cSlaveConfigurationGetAddress.....	10-19
ni845xI2cSlaveConfigurationGetAutoRemovalTimeout.....	10-20
ni845xI2cSlaveConfigurationGetCommandBufferSize.....	10-21
ni845xI2cSlaveConfigurationGetDataBufferSize.....	10-22
ni845xI2cSlaveConfigurationOpen.....	10-23
ni845xI2cSlaveConfigurationSetAddress.....	10-24
ni845xI2cSlaveConfigurationSetAutoRemovalTimeout.....	10-25
ni845xI2cSlaveConfigurationSetCommandBufferSize.....	10-26
ni845xI2cSlaveConfigurationSetDataBufferSize.....	10-27
Slave.....	10-28
ni845xI2cSlaveGetWriteInfo .....	10-28
ni845xI2cSlaveRead.....	10-29
ni845xI2cSlaveStart .....	10-31
ni845xI2cSlaveStop.....	10-32
ni845xI2cSlaveWaitForEvent .....	10-33
ni845xI2cSlaveWrite.....	10-36

## Chapter 11

### Using the NI-845x SPI API

NI-845x SPI Basic Programming Model.....	11-1
SPI Configure .....	11-2
SPI Write Read.....	11-2
SPI Timing Characteristics.....	11-2
NI-845x SPI Scripting Programming Model.....	11-3
Script: Enable SPI.....	11-5
Script: Configure Phase, Polarity, Clock Rate, Number of Bits .....	11-5
Script: Chip Select Low.....	11-5
Script: Write Read.....	11-5
Script: Chip Select High.....	11-6
Script: Disable SPI .....	11-6
Run Script.....	11-6
Extract Read Data.....	11-6

## Chapter 12

### NI-845x SPI API for LabVIEW

General Device .....	12-2
NI-845x Close Reference.vi .....	12-2
NI-845x Device Property Node.....	12-4
NI-845x Device Reference.....	12-7

Configuration.....	12-8
NI-845x SPI Configuration Property Node .....	12-8
NI-845x SPI Create Configuration Reference.vi .....	12-11
Basic .....	12-13
NI-845x SPI Write Read.vi .....	12-13
Scripting .....	12-16
NI-845x SPI Create Script Reference.vi .....	12-16
NI-845x SPI Extract Script Read Data.vi .....	12-18
NI-845x SPI Run Script.vi.....	12-20
NI-845x SPI Script Clock Polarity Phase.vi .....	12-22
NI-845x SPI Script Clock Rate.vi.....	12-24
NI-845x SPI Script CS High.vi.....	12-26
NI-845x SPI Script CS Low.vi .....	12-28
NI-845x SPI Script Delay (Microsecond).vi.....	12-30
NI-845x SPI Script Delay (Millisecond).vi .....	12-32
NI-845x SPI Script DIO Configure Line.vi .....	12-34
NI-845x SPI Script DIO Configure Port.vi.....	12-36
NI-845x SPI Script DIO Read Line.vi .....	12-38
NI-845x SPI Script DIO Read Port.vi.....	12-40
NI-845x SPI Script DIO Write Line.vi .....	12-42
NI-845x SPI Script DIO Write Port.vi.....	12-44
NI-845x SPI Script Disable SPI.vi.....	12-46
NI-845x SPI Script Enable SPI.vi.....	12-48
NI-845x SPI Script Num Bits Per Sample.vi.....	12-50
NI-845x SPI Script Write Read.vi .....	12-52

## Chapter 13

### NI-845x SPI API for C

Section Headings .....	13-1
Purpose.....	13-1
Format .....	13-1
Inputs and Outputs .....	13-1
Description .....	13-1
Data Types.....	13-1
List of Functions.....	13-2
General Device .....	13-8
ni845xClose .....	13-8
ni845xCloseFindDeviceHandle .....	13-9
ni845xDeviceLock .....	13-10
ni845xDeviceUnlock .....	13-11
ni845xFindDevice.....	13-12
ni845xFindDeviceNext .....	13-14
ni845xOpen.....	13-15

ni845xSetIoVoltageLevel .....	13-16
ni845xSetTimeout .....	13-17
ni845xStatusToString .....	13-18
Configuration .....	13-20
ni845xSpiConfigurationClose .....	13-20
ni845xSpiConfigurationGetChipSelect .....	13-21
ni845xSpiConfigurationGetClockPhase .....	13-22
ni845xSpiConfigurationGetClockPolarity .....	13-23
ni845xSpiConfigurationGetClockRate .....	13-24
ni845xSpiConfigurationGetNumBitsPerSample .....	13-25
ni845xSpiConfigurationGetPort .....	13-26
ni845xSpiConfigurationOpen .....	13-27
ni845xSpiConfigurationSetChipSelect .....	13-28
ni845xSpiConfigurationSetClockPhase .....	13-29
ni845xSpiConfigurationSetClockPolarity .....	13-30
ni845xSpiConfigurationSetClockRate .....	13-31
ni845xSpiConfigurationSetNumBitsPerSample .....	13-32
ni845xSpiConfigurationSetPort .....	13-33
Basic .....	13-34
ni845xSpiWriteRead .....	13-34
Scripting .....	13-36
ni845xSpiScriptClockPolarityPhase .....	13-36
ni845xSpiScriptClockRate .....	13-38
ni845xSpiScriptClose .....	13-39
ni845xSpiScriptCSHigh .....	13-40
ni845xSpiScriptCSLow .....	13-41
ni845xSpiScriptDelay .....	13-42
ni845xSpiScriptDioConfigureLine .....	13-43
ni845xSpiScriptDioConfigurePort .....	13-44
ni845xSpiScriptDioReadLine .....	13-45
ni845xSpiScriptDioReadPort .....	13-47
ni845xSpiScriptDioWriteLine .....	13-48
ni845xSpiScriptDioWritePort .....	13-50
ni845xSpiScriptDisableSPI .....	13-51
ni845xSpiScriptEnableSPI .....	13-52
ni845xSpiScriptExtractReadData .....	13-53
ni845xSpiScriptExtractReadDataSize .....	13-54
ni845xSpiScriptNumBitsPerSample .....	13-55
ni845xSpiScriptOpen .....	13-56
ni845xSpiScriptReset .....	13-57
ni845xSpiScriptRun .....	13-58
ni845xSpiScriptUsDelay .....	13-59
ni845xSpiScriptWriteRead .....	13-60

## Chapter 14

### Using the NI-845x SPI Stream API

NI-845x SPI Stream Programming Model .....	14-1
SPI Stream Configure .....	14-2
SPI Stream Start .....	14-2
SPI Stream Read .....	14-2
SPI Stream Stop .....	14-2
Waveform 1 .....	14-3
Extra SPI Pin Descriptions .....	14-4
CONV .....	14-4
DRDY .....	14-4
Chip Select .....	14-4

## Chapter 15

### NI-845x SPI Stream API for LabVIEW

General Device .....	15-2
NI-845x Close Reference.vi .....	15-2
NI-845x Device Property Node .....	15-4
NI-845x Device Reference .....	15-7
Configuration .....	15-8
NI-845x SPI Stream Configuration Property Node .....	15-8
NI-845x SPI Stream Create Configuration Reference.vi .....	15-16
Streaming .....	15-18
NI-845x SPI Stream Read.vi .....	15-18
NI-845x SPI Stream Start.vi .....	15-20
NI-845x SPI Stream Stop.vi .....	15-22

## Chapter 16

### NI-845x SPI Stream API for C

Section Headings .....	16-1
Purpose .....	16-1
Format .....	16-1
Inputs and Outputs .....	16-1
Description .....	16-1
Data Types .....	16-1
List of Functions .....	16-2
General Device .....	16-5
ni845xClose .....	16-5
ni845xCloseFindDeviceHandle .....	16-6
ni845xDeviceLock .....	16-7
ni845xDeviceUnlock .....	16-8



ni845xFindDevice .....	16-9
ni845xFindDeviceNext.....	16-11
ni845xOpen .....	16-12
ni845xSetTimeout .....	16-13
ni845xStatusToString .....	16-14
Configuration .....	16-16
ni845xSpiStreamConfigurationClose.....	16-16
ni845xSpiStreamConfigurationOpen .....	16-17
ni845xSpiStreamConfigurationGetNumBits.....	16-18
ni845xSpiStreamConfigurationGetNumSamples.....	16-19
ni845xSpiStreamConfigurationGetPacketSize.....	16-20
ni845xSpiStreamConfigurationGetClockPhase .....	16-21
ni845xSpiStreamConfigurationWave1GetPinConfig .....	16-22
ni845xSpiStreamConfigurationGetClockPolarity .....	16-23
ni845xSpiStreamConfigurationWave1GetTimingParam.....	16-24
ni845xSpiStreamConfigurationWave1SetMosiData.....	16-26
ni845xSpiStreamConfigurationSetNumBits .....	16-28
ni845xSpiStreamConfigurationSetNumSamples .....	16-29
ni845xSpiStreamConfigurationSetPacketSize .....	16-30
ni845xSpiStreamConfigurationSetClockPhase.....	16-31
ni845xSpiStreamConfigurationWave1SetPinConfig .....	16-32
ni845xSpiStreamConfigurationSetClockPolarity.....	16-33
ni845xSpiStreamConfigurationWave1SetTimingParam .....	16-34
Streaming .....	16-36
ni845xSpiStreamRead .....	16-36
ni845xSpiStreamStart.....	16-38
ni845xSpiStreamStop .....	16-39

## Chapter 17

### Using the NI-845x DIO API

NI-845x DIO Basic Programming Model .....	17-1
DIO Port Configure .....	17-2
DIO Port Write .....	17-2
DIO Port Read .....	17-2
DIO Line Write.....	17-2
DIO Line Read .....	17-2

## Chapter 18

### NI-845x DIO API for LabVIEW

General Device .....	18-2
NI-845x Close Reference.vi.....	18-2
NI-845x Device Property Node .....	18-4
NI-845x Device Reference.....	18-7
Basic .....	18-8
NI-845x DIO Read Line.vi .....	18-8
NI-845x DIO Read Port.vi.....	18-10
NI-845x DIO Write Line.vi .....	18-12
NI-845x DIO Write Port.vi.....	18-14

## Chapter 19

### NI-845x DIO API for C

Section Headings .....	19-1
Purpose.....	19-1
Format .....	19-1
Inputs and Outputs .....	19-1
Description .....	19-1
Data Types.....	19-1
List of Functions.....	19-2
General Device .....	19-4
ni845xClose .....	19-4
ni845xCloseFindDeviceHandle .....	19-5
ni845xDeviceLock .....	19-6
ni845xDeviceUnlock .....	19-7
ni845xFindDevice.....	19-8
ni845xFindDeviceNext .....	19-10
ni845xOpen.....	19-11
ni845xSetIoVoltageLevel .....	19-12
ni845xSetTimeout.....	19-13
ni845xStatusToString.....	19-14
Basic .....	19-16
ni845xDioReadLine .....	19-16
ni845xDioReadPort.....	19-18
ni845xDioSetPortLineDirectionMap.....	19-19
ni845xDioSetDriverType.....	19-20
ni845xDioWriteLine .....	19-21
ni845xDioWritePort.....	19-22

**Appendix A**  
**NI USB-845x Hardware Specifications**

**Appendix B**  
**Technical Support and Professional Services**

**Glossary**

**Index**

# About This Manual

---

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

Use this manual to learn the basics of I<sup>2</sup>C and SPI communication with NI-845x, 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.

---

# Introduction

This chapter introduces the Inter-IC (I<sup>2</sup>C) and Serial Peripheral Interface (SPI) buses.

## I<sup>2</sup>C Bus

---

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

## I<sup>2</sup>C Terminology

This manual uses the following I<sup>2</sup>C bus terms:

I <sup>2</sup> C	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 <sup>2</sup> C 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<sup>2</sup>C Bus

The I<sup>2</sup>C bus is a two-wire half-duplex serial interface. The two wires, SDA and SCL, are both bidirectional. The I<sup>2</sup>C 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<sup>2</sup>C bus has a unique 7-bit I<sup>2</sup>C 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<sup>2</sup>C 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.

I<sup>2</sup>C 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<sup>2</sup>C defines an arbitration mechanism to determine which master gets control of the bus.

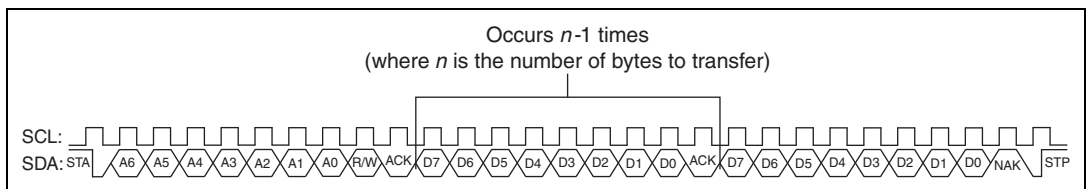
## I<sup>2</sup>C Arbitration

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<sup>2</sup>C bus has an arbitration scheme to guarantee no data corruption.

With I<sup>2</sup>C, 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<sup>2</sup>C bus so the arbitration is finalized once the entire master code has been transferred.

## I<sup>2</sup>C Transfers



**Figure 1-1.** I<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C ACK Polling

In some cases when the master sends an address, a slave may not immediately acknowledge (ACK) its address. The slave may be busy and thus temporarily detached from the bus. Typically EEPROMs do this while storing a page of data. Once the write cycle completes, they reattach to the I<sup>2</sup>C bus.

When ACK polling, the master repeatedly resends the start, address, and direction bit to the slave until the address is acknowledged. Once the address is acknowledged, the data transfer continues normally.

## I<sup>2</sup>C Clock Stretching

Because the master controls the clock, the I<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C Extended (10-Bit) Addressing

Typical I<sup>2</sup>C devices use a 7-bit addressing scheme. I<sup>2</sup>C also defines a 10-bit addressing scheme that allows up to 1024 additional addresses to be connected to the I<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C bus; however, the NXP I<sup>2</sup>C specification describes master code 0 as reserved for test and diagnostic purposes.

## I<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C 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<sup>2</sup>C as  $0.7 * V_{DD}$ . On SMBus, logic high is defined as 2.1 V.

Logic low is defined on I<sup>2</sup>C as  $0.3 * V_{DD}$ . On SMBus, logic low is defined as 0.8 V.

## Current Levels

The sink current also varies between I<sup>2</sup>C and SMBus. In I<sup>2</sup>C, 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  $\mu$ A. This determines the lowest acceptable value of the pull-up resistor. At 3 V in Standard mode, an I<sup>2</sup>C bus should have a pull-up of > 1 k $\Omega$ ; SMBus should have a pull-up of > 8.5 k $\Omega$ . However, many SMBus systems violate this rule; a common range for both SMBus and I<sup>2</sup>C tends to be in the 2.4–3.9 k $\Omega$  range, but may vary significantly for various speeds and bus capacitance ranges.

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

Throughout this document, we will refer to the bus as an I<sup>2</sup>C bus. For information about compatibility of your NI 845x 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.

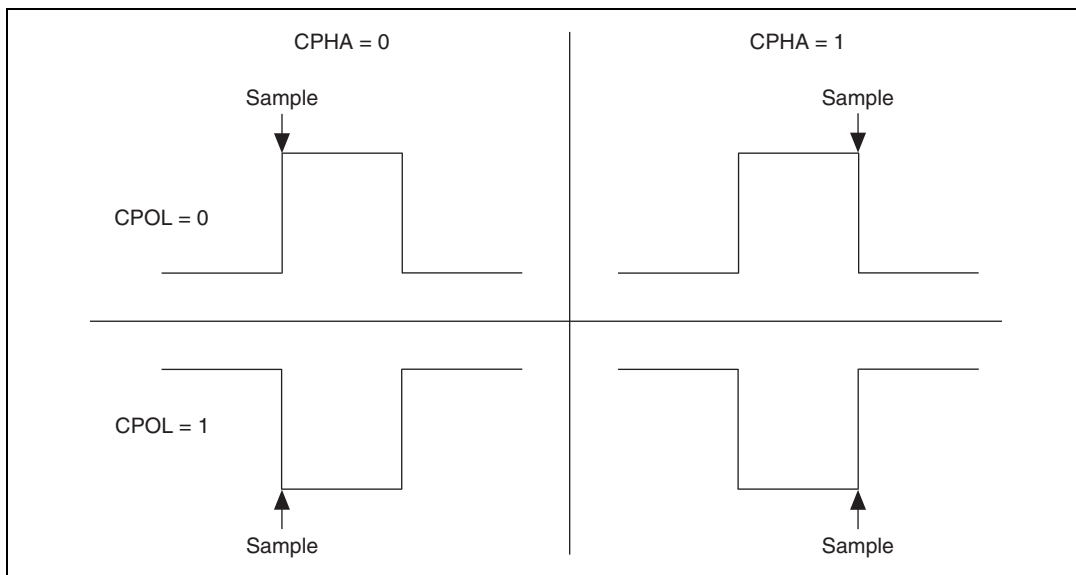
## Clock and Polarity

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<sup>2</sup>C, 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.

---

# Installation

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

## Software Installation

---

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



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

1. Insert the *NI-845x Software* installation media into your CD-ROM or DVD-ROM drive. The installer launches if your optical drive plays data disks automatically. If the installer does not launch automatically, navigate to the media using Windows Explorer and launch the `autorun.exe` file from your *NI-845x Software* installation media.
2. The Installation Wizard guides you through the necessary steps to install the NI-845x 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.

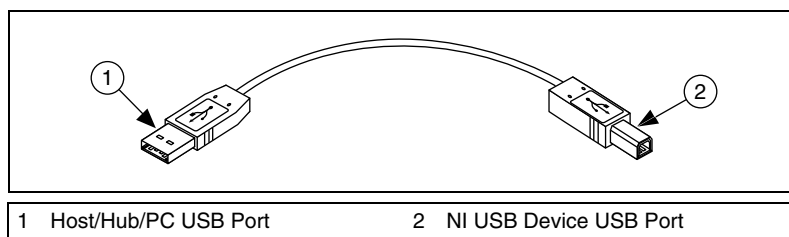
Store the device in the antistatic package when the device is not in use.

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.



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](http://ni.com.support/install) for troubleshooting information.

# System Configuration API

---

NI-845x supports the National Instruments System Configuration API, which provides programmatic access to many operations in MAX. This enables you to perform these operations within your application.

The System Configuration API gathers information using various product experts. You can create a filter to gather information for one type of product such as filtering for NI-845x devices only. The NI-845x expert programmatic name is *845x*.

---

# NI USB-845x Hardware Overview

## Overview

---

NI USB-845x modules are USB 2.0 devices that provide I<sup>2</sup>C 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<sup>2</sup>C (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

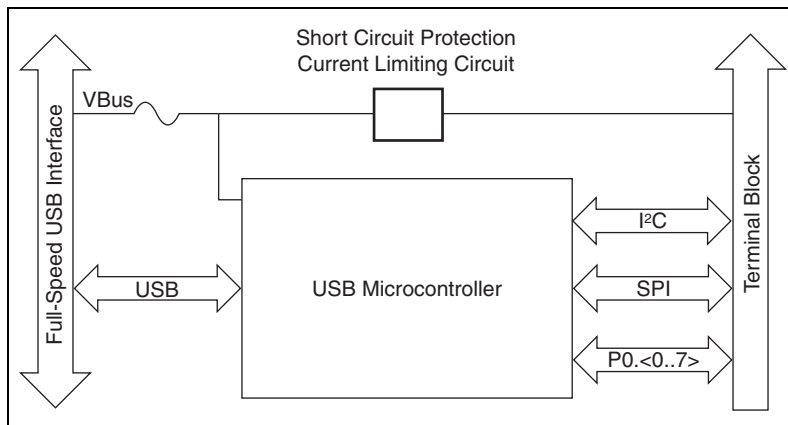


Figure 3-1. NI USB-8451 Block Diagram

## 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

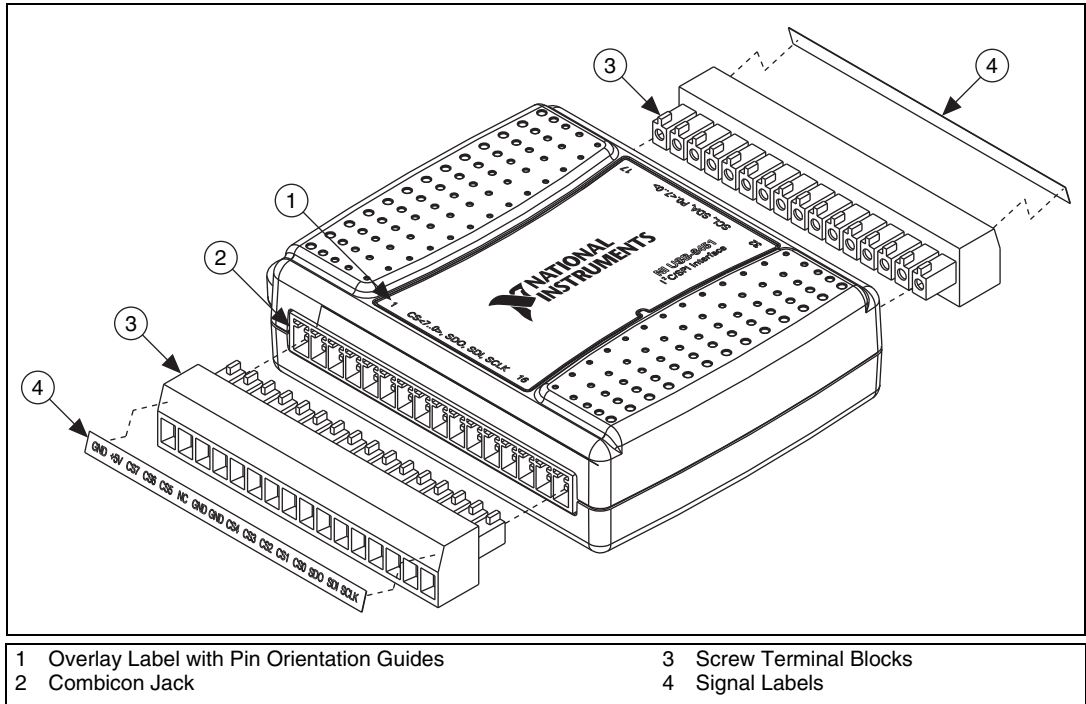
Complete the following steps to set up the hardware:

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.



**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 <sup>2</sup> C SDA
	14	SPI MOSI (SDO)		30	I <sup>2</sup> C 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	<b>Chip Select Signals</b> —Outputs used to select the desired SPI peripheral device.
SPI MOSI (SDO)	Output	<b>Master Output Slave Input</b> —SPI communication signal to slave device.
SPI MISO (SDI)	Input	<b>Master Input Slave Output</b> —SPI communication signal from slave device.
SPI CLK (SCLK)	Output	<b>SPI Clock</b> —SPI output clock signal to slave devices capable of clock rates up to 12 MHz.
I <sup>2</sup> C SDA	Open-drain	<b>I<sup>2</sup>C Serial Data</b> —Data signal for I <sup>2</sup> C communication.
I <sup>2</sup> C SCL	Open-drain	<b>I<sup>2</sup>C Clock</b> —I <sup>2</sup> C clock signal to slave devices capable of clock rates up to 250 kHz.
P0.<0..7>	Input or output	<b>Digital I/O Signals</b> —You can individually configure each signal as an input or output. You can configure the port for open-drain or push-pull output. <sup>1</sup>
+5 V	Output	<b>+5 V</b> —The voltage source provided by the USB host. The voltage is nominally 5 V, but varies from system to system.
GND	—	<b>Ground</b> —The reference for the digital signals and the +5 VDC supply.
NC	—	<b>No Connect</b> —Do not connect any signals to this terminal.

<sup>1</sup> If you configure the DIO port for open-drain output, you must supply pull-up resistors to V<sub>cc</sub> (3.3 or 5 V). The resistor value must not be lower than 1 kΩ.

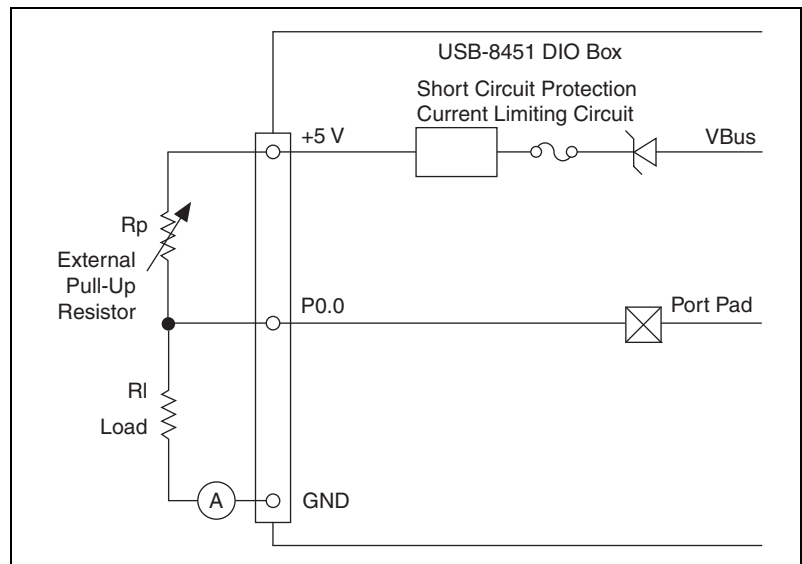
## Front-End I/O Interfaces

### Digital I/O (DIO)

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

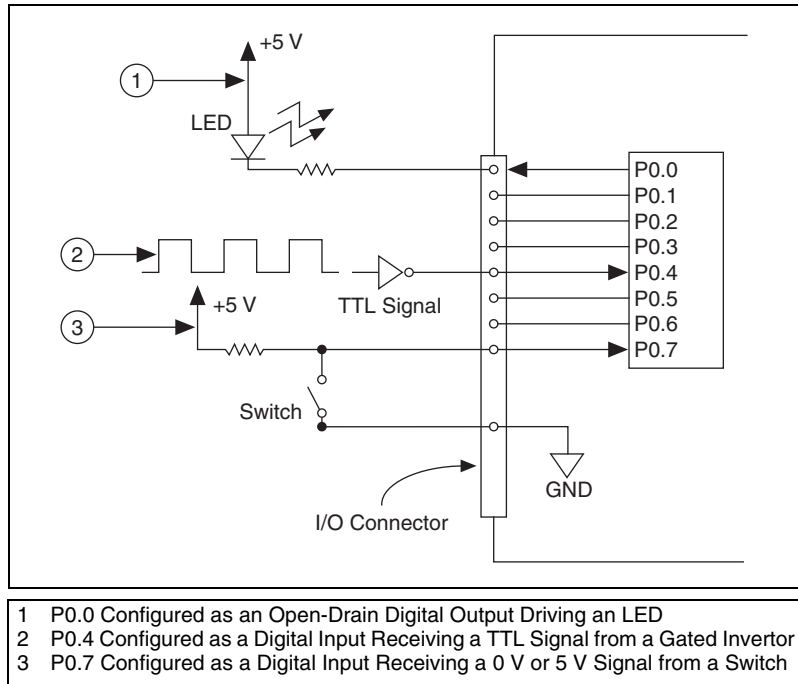
You can program each DIO line individually as a static DI or DO line. You can use static DIO lines to monitor or control digital signals. All samples of static DI lines and updates of DO lines are software timed.

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 ( $R_p$ ), as shown in Figure 3-3. Do not use a pull-up resistor of less than 1 k $\Omega$ .



**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.



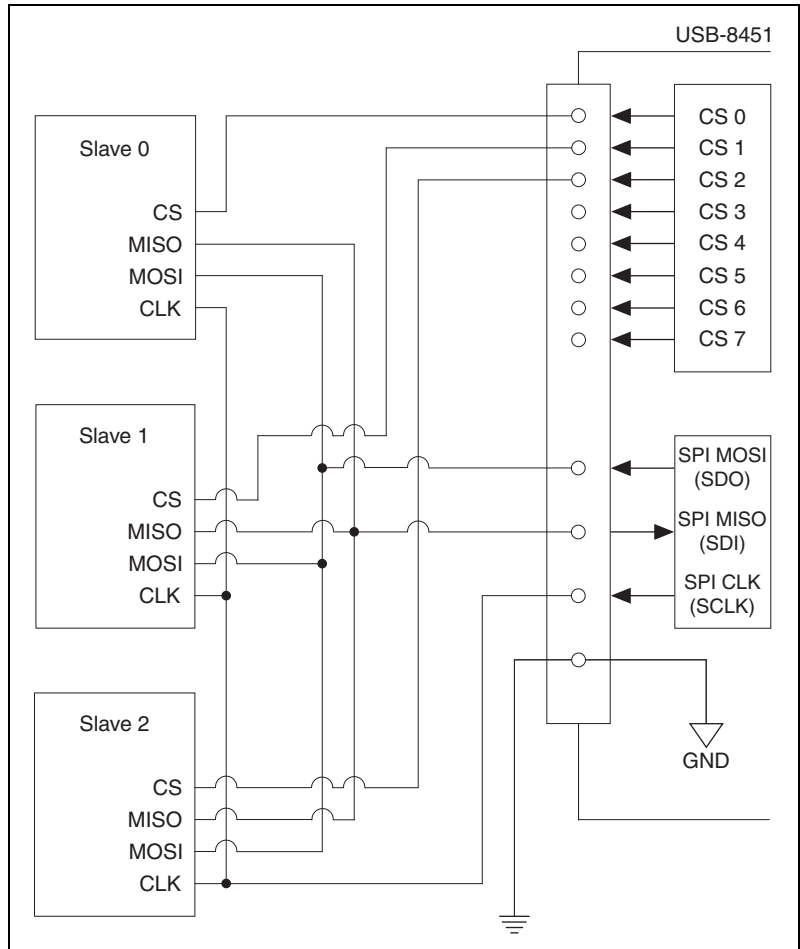
**Figure 3-4.** Example of Connecting a Load



**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.

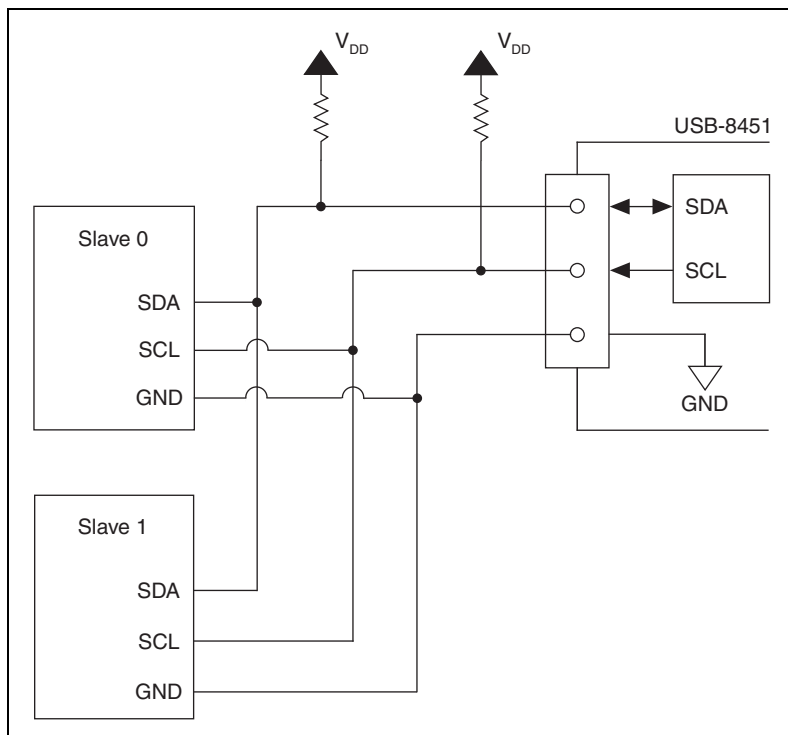


**Figure 3-5.** SPI Interface to Three Peripherals



## I<sup>2</sup>C Interface

Figure 3-6 shows a typical I<sup>2</sup>C interface to two peripherals. All devices on the I<sup>2</sup>C bus share the SDA and SCL signals. SDA and SCL must be pulled up externally. Refer to the I<sup>2</sup>C specification to select the correct resistor values for your bus.



**Figure 3-6.** I<sup>2</sup>C 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<sup>2</sup>C 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 and SPI CS 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<sup>2</sup>C (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<sup>2</sup>C, SPI, and DIO at multiple logic levels.

The NI USB-8452 is available in an enclosure and as a board-only version. In this manual, the enclosure version (shown in Figure 3-7) is referred to as the NI USB-8452, and the board-only version is referred to as the

NI USB-8452 OEM. They share the same electrical design and pinout at the connector. Unless otherwise noted, all information in this manual applies to both the NI USB-8452 and NI USB-8452 OEM.

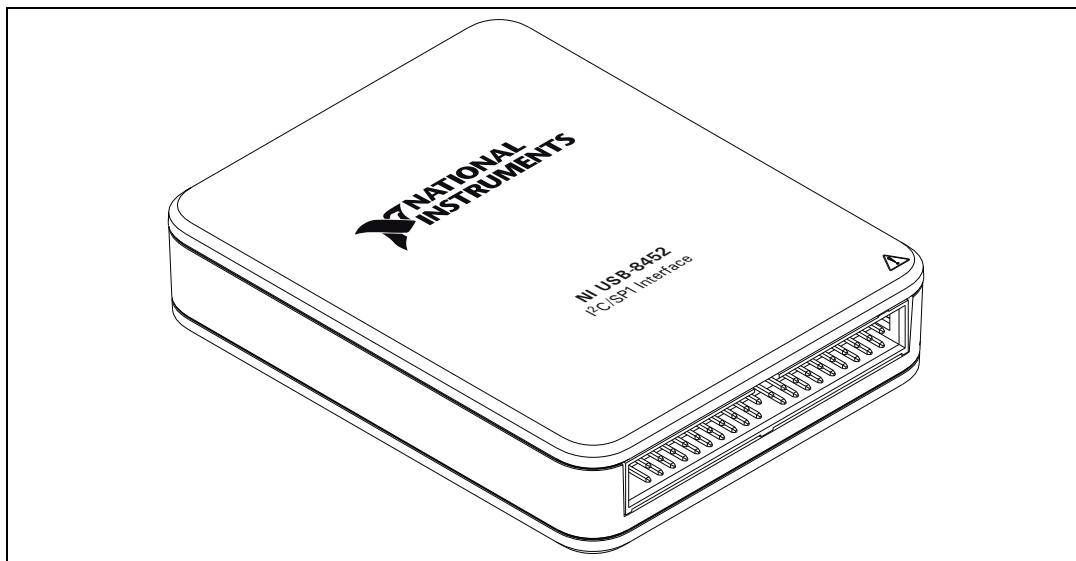


Figure 3-7. NI USB-8452 with Enclosure

## Block Diagram

The block diagram in Figure 3-8 shows key NI USB-8452 module functional components.

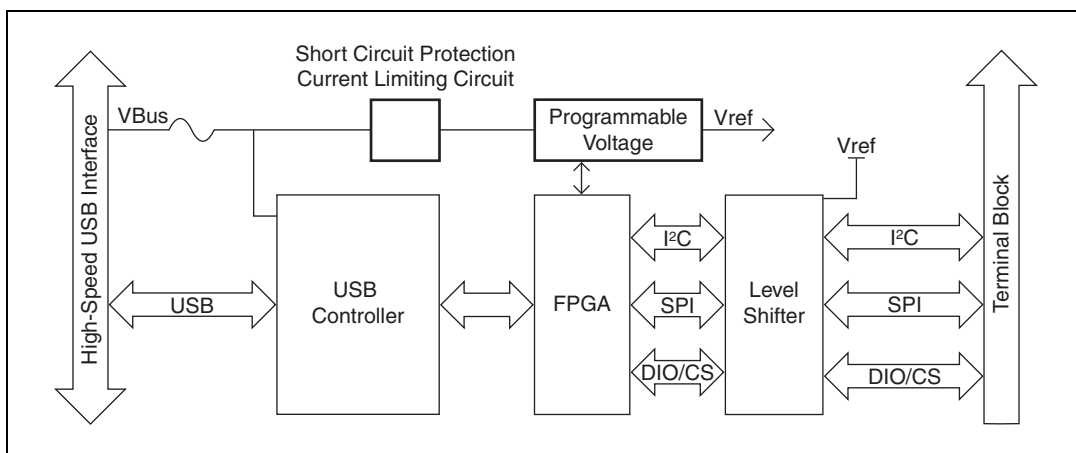


Figure 3-8. NI USB-8452 Block Diagram

The NI USB-8452 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 supports SPI data acquisition up to 50 MHz and I<sup>2</sup>C communication up to 3.3 MHz. The programmable reference voltage covers logic families from 1.2 V to 3.3 V, which makes the NI USB-8452 versatile for most SPI/I<sup>2</sup>C 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. Refer to the *NI-845x Software and Hardware Installation Guide* for more information.

## Setting Up Hardware

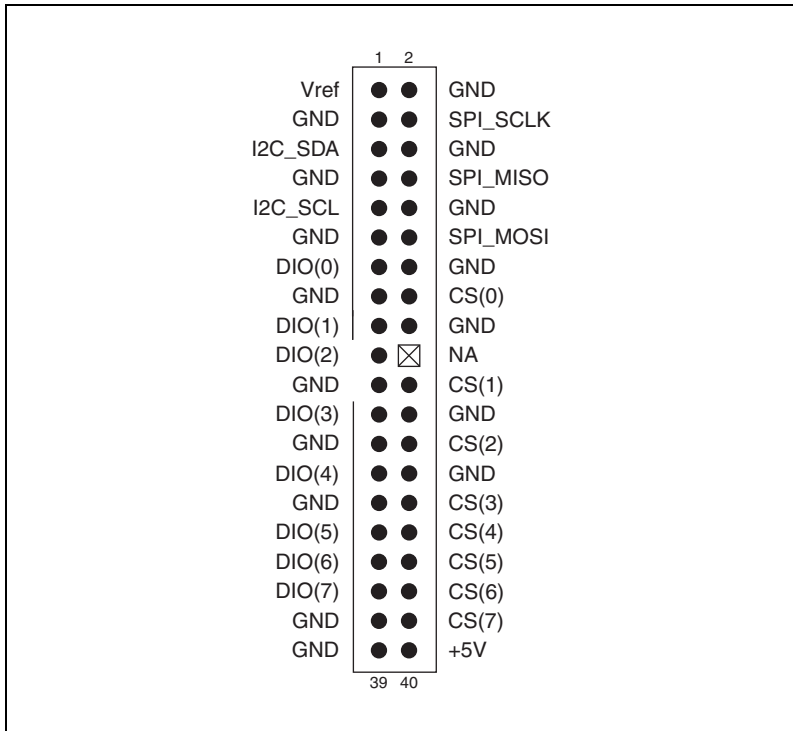
Complete the following steps to set up the hardware:

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



**Note** You can use a standard 40-pin IDE (ribbon) cable to access the front-end I/O pins (SPI, I<sup>2</sup>C, and digital I/O) of the NI USB-8452 module. We also provide a fly-lead cable (part number 152580-0R25) as an accessory for easy wiring.

2. Connect the other end of the cable to your board. Refer to Figure 3-9 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 3-9.** IDE-40 Connector Pin Assignments



**Note** The above pinout applies to both the enclosure and OEM versions.

- The USB-8452 has a USB series B-type receptacle. Use the USB cable included in the kit (enclosure version only) to connect to the host PC.

## 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	<b>SPI Clock</b> —SPI output clock signal to slave devices capable of clock rates up to 50 MHz.
SPI_MOSI	Output	<b>Master Output Slave Input</b> —SPI communication signal to slave device.

**Table 3-4.** Signal Descriptions

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

<sup>1</sup> You can configure CS(0) as hardware-timed chip-select, which has a fixed timing relationship to SPI signal lines. Refer to Chapter 14, *Using the NI-845x SPI Stream API*, for details.

<sup>2</sup> Some of these pins have special functionality in SPI stream mode. Refer to Chapter 14, *Using the NI-845x SPI Stream API*, for details.

<sup>3</sup> You can enable or disable onboard pull-up resistors. You must enable these for Vref  $\leq$  1.8 V for the FPGA to properly detect a low-to-high transition. Refer to Chapter 5, *Using the NI-845x I<sup>2</sup>C API*, for more information about enabling pull-ups on the I<sup>2</sup>C lines.

## Front-End I/O Interfaces

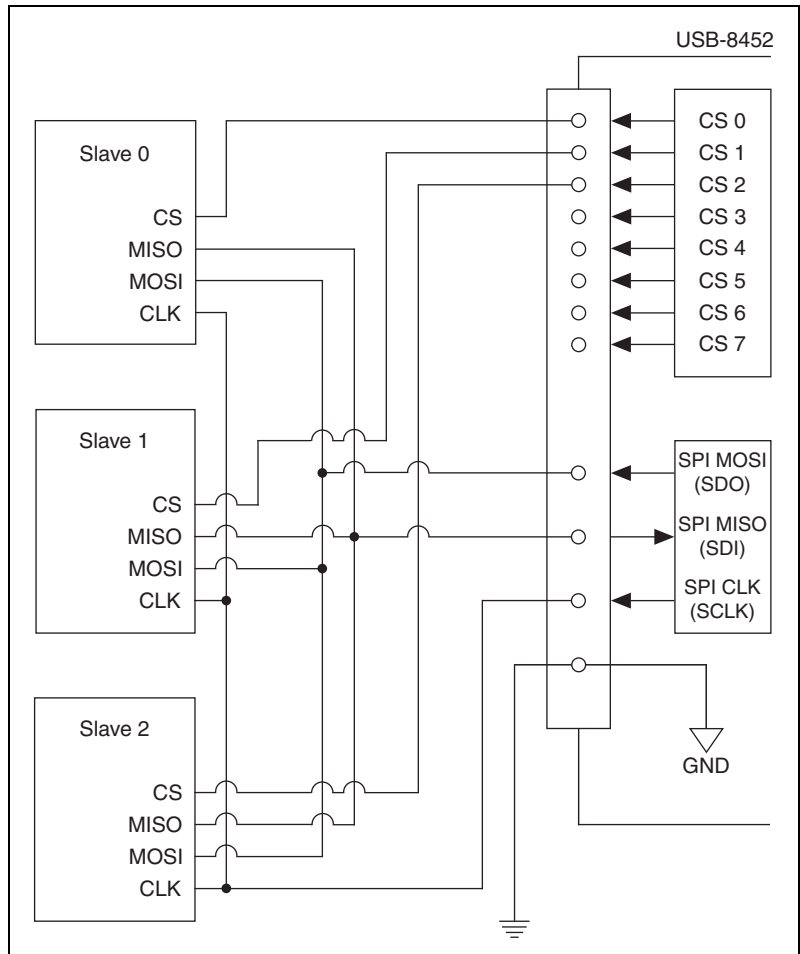


**Caution** Exceeding the maximum input voltage ratings or maximum output ratings, which are listed in Appendix A, *NI USB-845x Hardware Specifications*, can damage the USB device and the computer. National Instruments is not liable for any damage resulting from such signal connections.

### SPI Interface

The NI USB-8452 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 supports logic families of 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V.

Figure 3-10 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 3-10.** SPI Interface to Three Peripherals

The NI USB-8452 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

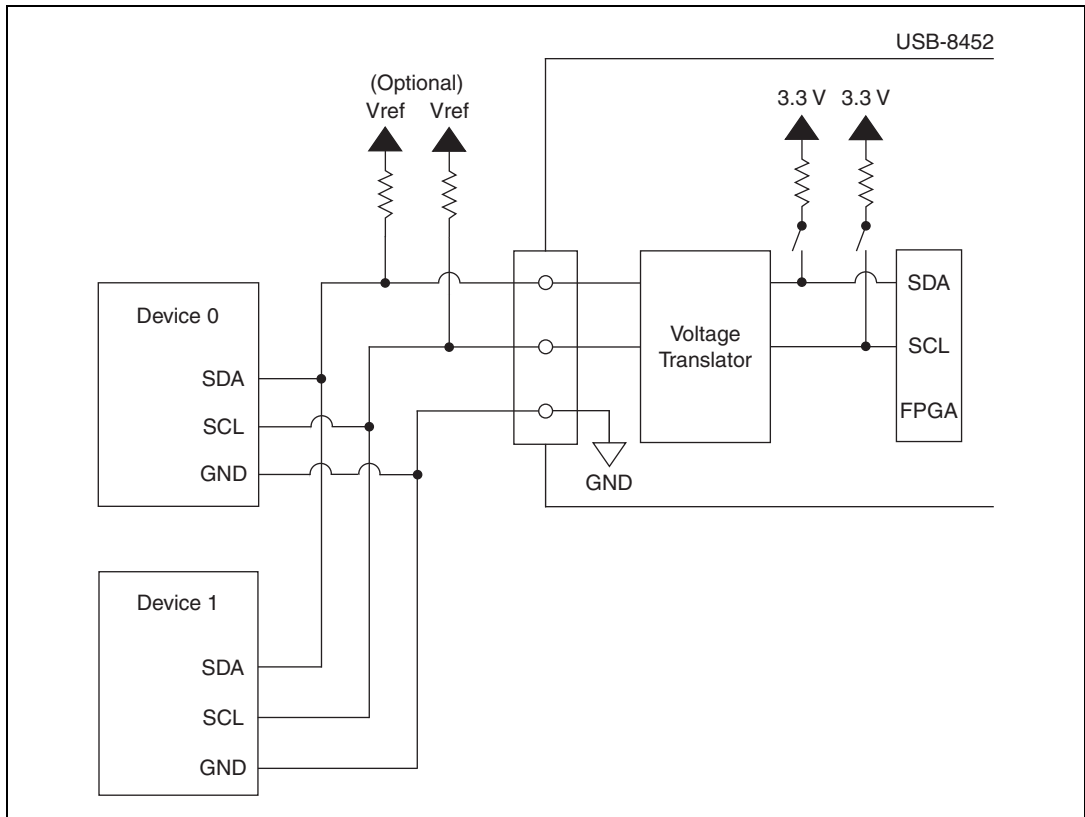
The SPI standard API provides the most fundamental SPI transaction type: write/read. You can access most existing SPI devices using this transaction. This mode is backward compatible with the NI USB-8451 and works with the NI-845x basic and scripting APIs.

## 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 14, [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<sup>2</sup>C Interface

You can configure the NI USB-8452 I<sup>2</sup>C Interface as either a master or a slave on an I<sup>2</sup>C bus. Figure 3-11 shows a typical I<sup>2</sup>C interface to two peripherals. All devices on the I<sup>2</sup>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 standards. SDA and SCL lines may be pulled up internally or externally for 2.5 V and 3.3 V. Refer to the I<sup>2</sup>C specification to select the proper resistor values if using external pull-ups.



**Figure 3-11.** I<sup>2</sup>C Interface to Two Peripherals

The NI USB-8452 I<sup>2</sup>C interface supports Standard mode, Fast mode, Fast mode Plus, and High Speed mode (HS mode), defined in I<sup>2</sup>C 4.0 specifications. Refer to the *I<sup>2</sup>C Interface* section in Appendix A, *NI USB-845x Hardware Specifications*, for list of supported I<sup>2</sup>C data rates.

Refer to Chapter 5, *Using the NI-845x I<sup>2</sup>C API*, for more information about programming and using the I<sup>2</sup>C interface.

## Digital I/O (DIO)

You can program each NI USB-8452 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 can also configure the logic level the same way as SPI and I<sup>2</sup>C 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 k $\Omega$ ).

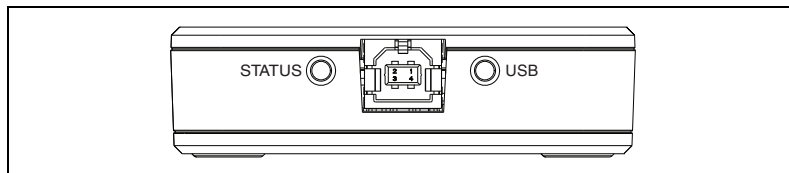
Refer to Chapter 17, *Using the NI-845x DIO API*, for more information about programming and using the DIO lines.



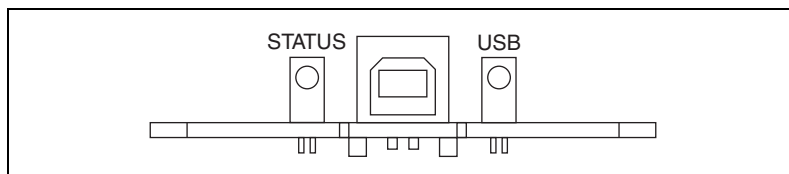
**Note** While the device is in USB suspend, all I/O outputs are disabled.

## LED Indicators

The NI USB-8452 has two LED indicators alongside the USB connector, as shown in Figures 3-12 and 3-13.



**Figure 3-12.** LED Indicators on NI USB-8452



**Figure 3-13.** LED Indicators on NI USB-8452 OEM

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<sup>2</sup>C interface's current working status.

State	Status
Off	SPI/I <sup>2</sup> C interface is idle
Blinking green	SPI/I <sup>2</sup> C 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.
- 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 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<sup>2</sup>C, SPI, DIO, and CS lines, among which SPI, DIO and CS lines are weakly pulled down to GND with 40 k $\Omega$  resistors. The NI USB-8452 does not drive any of the signals high or low.

## Power Sources

### +5 V Power Source

The NI USB-8452 offers a 5 V output from pin 40. The voltage source is generated from an onboard regulator, with  $\pm 5\%$  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-845x 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-845x 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 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 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 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 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 k $\Omega$ ) to GND.

---

# Using the NI-845x API

The NI-845x 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 845x device, you first must create a device handle by providing the name of the NI 845x device configured in Measurement & Automation Explorer (MAX). After creating the device handle, the NI-845x software functions use the returned handle to determine which NI 845x device to communicate with.

The NI-845x API has other handles also. An example is a configuration handle that describes the device characteristics used for communication. An I<sup>2</sup>C 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-845x 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.

# Using the NI-845x I<sup>2</sup>C API

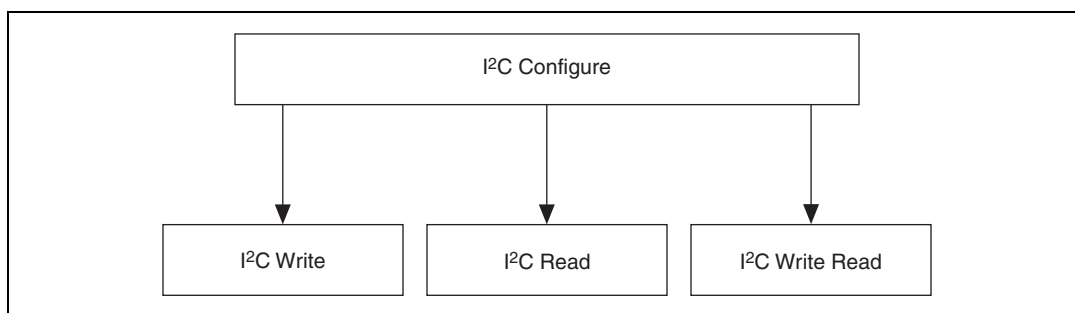
This chapter helps you get started with the I<sup>2</sup>C API.

## I<sup>2</sup>C Basic Programming Model

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

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

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



**Figure 5-1.** Basic Programming Model for I<sup>2</sup>C Communication

## I<sup>2</sup>C Configure

Use the **NI-845x I<sup>2</sup>C Configuration Property Node** in LabVIEW and `ni845xI2cConfiguration*` calls in other languages to set the specific I<sup>2</sup>C configuration that describes the characteristics of the device to communicate with.

## I<sup>2</sup>C Write

Use **NI-845x I<sup>2</sup>C Write.vi** in LabVIEW and `ni845xI2cWrite` in other languages to write an array of data to an I<sup>2</sup>C slave device.

## I<sup>2</sup>C Read

Use **NI-845x I<sup>2</sup>C Read.vi** in LabVIEW and `ni845xI2cRead` in other languages to read an array of data from an I<sup>2</sup>C slave device.

## I<sup>2</sup>C Write Read

Use **NI-845x I<sup>2</sup>C Write Read.vi** in LabVIEW and `ni845xI2cWriteRead` in other languages to write an array of data followed by a read (combined format) on an I<sup>2</sup>C slave device.

## I<sup>2</sup>C Scripting Programming Model

---

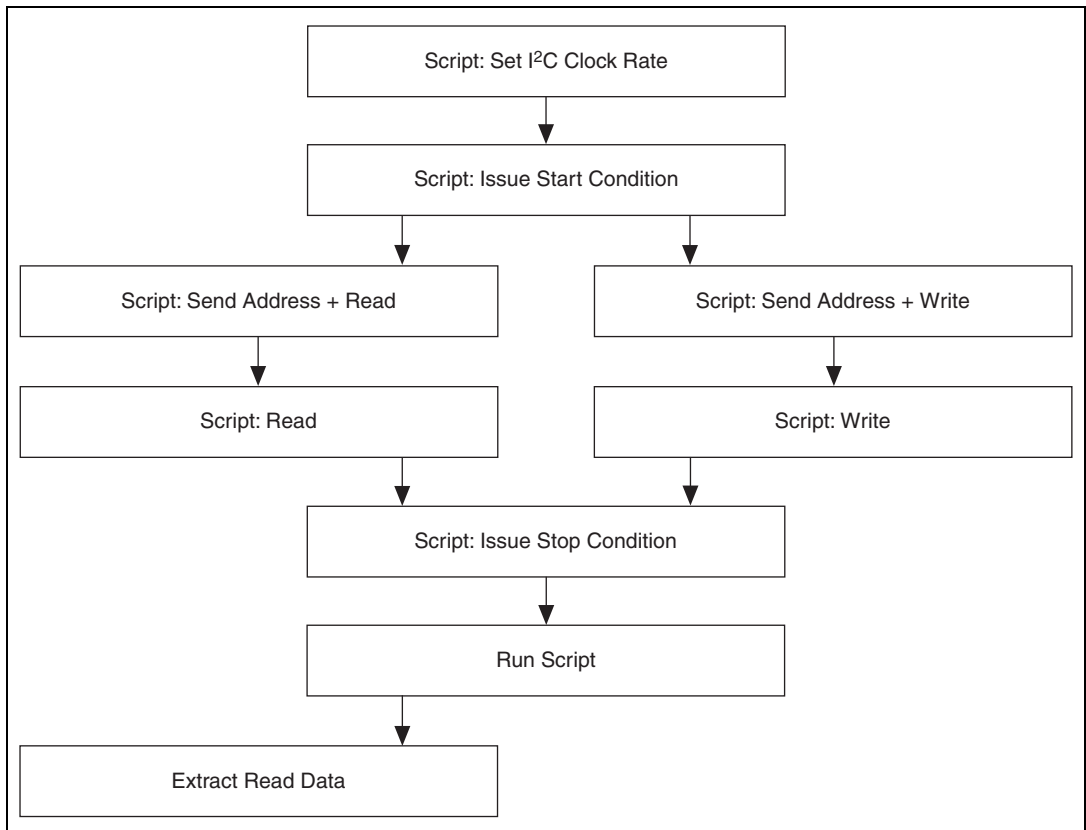
The NXP I<sup>2</sup>C specification is extremely flexible and allows multiple possibilities for constructing transactions beyond those handled by the I<sup>2</sup>C Basic API. The I<sup>2</sup>C Scripting API provides a set of script commands that allow you great flexibility in creating custom I<sup>2</sup>C 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 I<sup>2</sup>C 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 845x DIO pins configured for input) at multiple points within the transaction.
- Using the NI 845x 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 I<sup>2</sup>C Scripting API, the first step is to create a script that describes the communication between an I<sup>2</sup>C master and an I<sup>2</sup>C 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, I<sup>2</sup>C Script Read, I<sup>2</sup>C Script DIO Read Port, and I<sup>2</sup>C 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-845x I<sup>2</sup>C Scripting API. The diagram is followed by a description of each step in the model.



**Figure 5-2.** Example of Scripting Programming Model with Scripting API for I<sup>2</sup>C Communication

## Script: Set I<sup>2</sup>C Clock Rate

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

## Script: Set I<sup>2</sup>C ACK Poll Timeout

Use **NI-845x I<sup>2</sup>C Script ACK Poll Timeout.vi** in LabVIEW and `ni845xI2cScriptAckPollTimeout` in other languages to add an I<sup>2</sup>C Script ACK Poll Timeout command to the I<sup>2</sup>C script. This command sets the I<sup>2</sup>C ACK Poll timeout in milliseconds. When this value is nonzero, the NI-845x device ACK polls when sending an address + direction byte.

## Script: Pullup Enable

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

## Script: Set I<sup>2</sup>C High Speed Clock Rate

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

## Script: Set I<sup>2</sup>C High Speed Enable

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

## Script: Issue Start Condition

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

## Script: Send High Speed Master Code

Use **NI-845x I<sup>2</sup>C Script Master Code.vi** in LabVIEW and `ni845xI2cScriptHsMasterCode` in other languages to add an I<sup>2</sup>C Script HS Master Code command to the I<sup>2</sup>C script. This command transmits the I<sup>2</sup>C High Speed master code. This command is valid only for NI 845x devices that support High Speed I<sup>2</sup>C.

## Script: Send Address + Read

Use **NI-845x I<sup>2</sup>C Script Address+Read.vi** in LabVIEW and `ni845xI2cScriptAddressRead` in other languages to add an I<sup>2</sup>C Script Address+Read command to the I<sup>2</sup>C script. This command writes a 7-bit address, followed by the direction bit set to read, to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you run the script. If the ACK poll timeout is nonzero, this command performs ACK polling.

## Script: Read

Use **NI-845x I<sup>2</sup>C Script Read.vi** in LabVIEW and `ni845xI2cScriptRead` in other languages to add an I<sup>2</sup>C Script Read command to the I<sup>2</sup>C script. This command reads an array of data from a device connected to the I<sup>2</sup>C port you specify when you run the script.

## Script: Send Address + Write

Use **NI-845x I<sup>2</sup>C Script Address+Write.vi** in LabVIEW and `ni845xI2cScriptAddressRead` in other languages to add an I<sup>2</sup>C Script Address+Write command to the I<sup>2</sup>C script. This command writes a 7-bit address, followed by the direction bit set to write, to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you run the script. If the ACK poll timeout is nonzero, this command performs ACK polling.

## Script: Write

Use **NI-845x I<sup>2</sup>C Script Write.vi** in LabVIEW and `ni845xI2cScriptWrite` in other languages to add an I<sup>2</sup>C Script Write command to the I<sup>2</sup>C Script. This command writes an array of data to an I<sup>2</sup>C slave device when you run the script.

## Script: Issue Stop Condition

Use **NI-845x I<sup>2</sup>C Script Issue Stop.vi** in LabVIEW and `ni845xI2cScriptIssueStop` in other languages to add an I<sup>2</sup>C Script Issue Stop command to the I<sup>2</sup>C script. This command issues a stop condition on the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you run the script.

## Run Script

Use **NI-845x I<sup>2</sup>C Run Script.vi** in LabVIEW and `ni845xI2cScriptRun` in other languages to execute an I<sup>2</sup>C script on the desired device.

## Extract Read Data

Use **NI-845x I<sup>2</sup>C Extract Script Read Data.vi** in LabVIEW and `ni845xI2cScriptExtractReadData` in other languages to extract the desired read data from an I<sup>2</sup>C script that has been previously run. Each I<sup>2</sup>C script read command (I<sup>2</sup>C Script Read, I<sup>2</sup>C Script DIO Read Port, I<sup>2</sup>C Script DIO Read Line) returns a script read index to be passed into the Extract Read Data function.

---

# NI-845x I<sup>2</sup>C API for LabVIEW

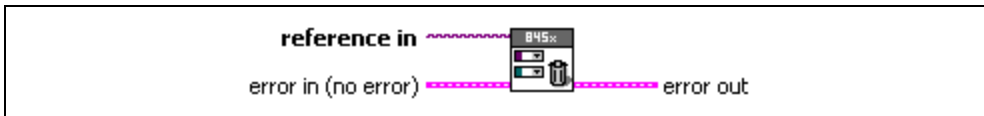
This chapter lists the LabVIEW VIs for the NI-845x I<sup>2</sup>C API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

# General Device

## NI-845x Close Reference.vi

### Purpose

Closes a previously opened reference.



### Inputs



**reference in** is a reference to an NI 845x device, I<sup>2</sup>C configuration, I<sup>2</sup>C Slave configuration, SPI configuration, SPI stream configuration, I<sup>2</sup>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.

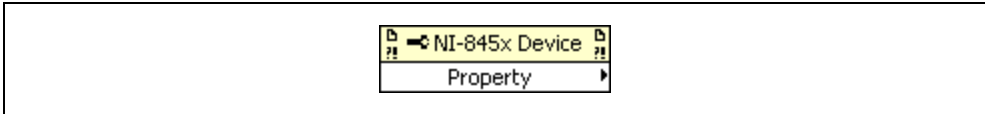
## Description

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

# NI-845x Device Property Node

## Purpose

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



## Inputs



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



### 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<sup>2</sup>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<sup>2</sup>C Pullup Enable

The **I<sup>2</sup>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.



### Timeout (ms)

The **Timeout (ms)** property sets the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.



**Note** It is highly recommended to set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

The default of this property is 30000 (30 seconds).

## NI-845x Device Reference

---

### Purpose

Specifies the device resource to be used for communication.



### Description

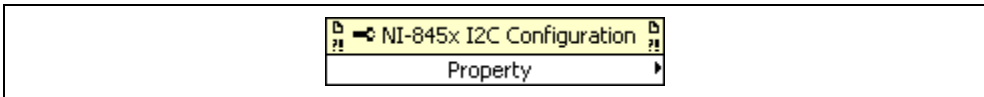
Use the **NI-845x Device Reference** to describe the NI 845x device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845x API call to invoke the function on the associated NI 845x device.

# Configuration

## NI-845x I<sup>2</sup>C Configuration Property Node

### Purpose

A property node with the NI-845x I<sup>2</sup>C Configuration class preselected. This property node allows you to query and modify I<sup>2</sup>C configuration properties of your NI 845x device.



### Inputs



**i2c configuration in** is a reference to a specific I<sup>2</sup>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<sup>2</sup>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 I<sup>2</sup>C Configuration Property Node**.



### Port

Specifies the I<sup>2</sup>C port that this configuration communicates across.

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

The default value of this property is 0.



### Clock Rate in kHz

Specifies the I<sup>2</sup>C clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845x 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<sup>2</sup>C slave device this configuration describes. **Address Size** uses the following values:

7 Bits

The NI 845x hardware uses the standard 7-bit addressing when communicating with the I<sup>2</sup>C slave device.

10 Bits

The NI 845x hardware uses the extended 10-bit addressing when communicating with the I<sup>2</sup>C slave device.

The default value of this property is 7 Bits.



### Address

Specifies the I<sup>2</sup>C slave address. The default address is 0. For 7-bit device addressing, the NXP I<sup>2</sup>C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845x 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-845x software, discard the direction bit and right-shift the byte value by one to create the 7-bit address.



### ACK Poll Timeout

Specifies the I<sup>2</sup>C ACK poll timeout in milliseconds. When this value is zero, ACK polling is disabled. Otherwise, the [NI-845x I2C Read.vi](#), [NI-845x I2C Write.vi](#), and [NI-845x I2C Write Read.vi](#) API calls ACK poll until an acknowledge (ACK) is detected or the timeout is reached.

ACK polling is not supported with 10-bit addressing. If **Address Size** is set to 10 bits, and ACK poll timeout is nonzero, an error is generated when attempting an I/O API call.

The default value of this property is 0 ms.



### HighSpeed:Enable

Enables High Speed (HS) mode. The default is set to High Speed mode disabled. When High Speed mode is enabled, the NXP I<sup>2</sup>C 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 845x device supports High Speed mode.



### HighSpeed:ClockRate

Specifies the I<sup>2</sup>C clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which High Speed clock rates your NI 845x 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 845x device supports.



### **HighSpeed:MasterCode**

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

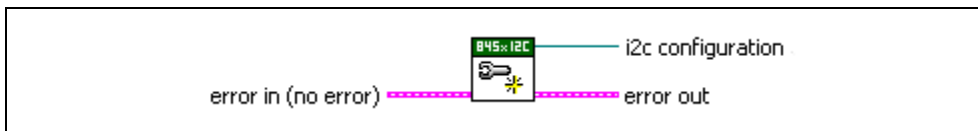
This property requires High Speed mode to be enabled.

The default value of this property is 1.

## NI-845x I<sup>2</sup>C Create Configuration Reference.vi

### Purpose

Creates a new NI-845x I<sup>2</sup>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<sup>2</sup>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

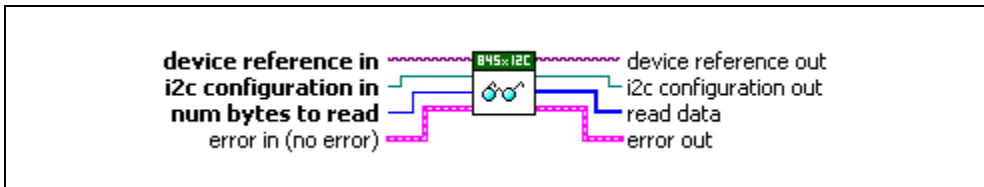
Use **NI-845x I<sup>2</sup>C Create Configuration Reference.vi** to create a new configuration to use with the NI-845x I<sup>2</sup>C Basic API. Pass the reference to a property node to make the configuration match the settings of your I<sup>2</sup>C slave. Then, pass the configuration to the I<sup>2</sup>C basic functions to execute them on the described I<sup>2</sup>C slave. After you finish communicating with your I<sup>2</sup>C 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 I<sup>2</sup>C Read.vi

### Purpose

Reads an array of data from an I<sup>2</sup>C slave device.



### Inputs



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



**i2c configuration in** is a reference to a specific I<sup>2</sup>C 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 I<sup>2</sup>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



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



**i2c configuration out** is a reference to the I<sup>2</sup>C configuration after this VI runs.



**read data** contains an array of read data from the I<sup>2</sup>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

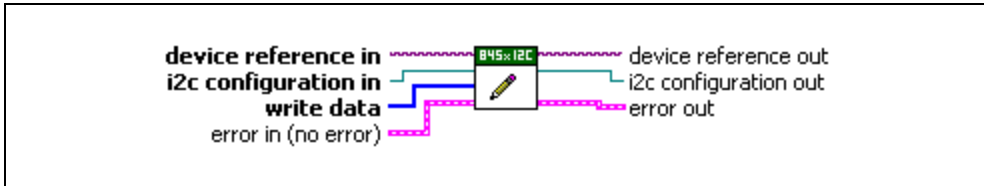
Use **NI-845x I<sup>2</sup>C Read.vi** to read an array of data from an I<sup>2</sup>C slave device. Per the NXP I<sup>2</sup>C Specification, each byte read up to the last byte is acknowledged. The last byte is not acknowledged. This VI first waits for the I<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7 or 10-bit I<sup>2</sup>C read transaction, per the NXP I<sup>2</sup>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 845x device tries to access the bus at the same time as another I<sup>2</sup>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, the NI 845x device ACK polls as specified in **i2c configuration in**. (Refer to the *I<sup>2</sup>C ACK Polling* section in Chapter 1, *Introduction*, for more information about ACK polling.) Otherwise, the transaction is completed, and a stop condition is generated per the NXP I<sup>2</sup>C Specification.

Before using **NI-845x I<sup>2</sup>C 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<sup>2</sup>C slave device.



### Inputs



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



**i2c configuration in** is a reference to a specific I<sup>2</sup>C 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<sup>2</sup>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



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



**i2c configuration out** is a reference to the I<sup>2</sup>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

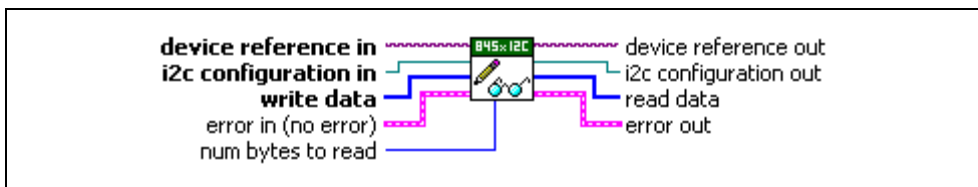
Use **NI-845x I2C Write.vi** to write an array of data to an I<sup>2</sup>C slave device. This VI first waits for the I<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7 or 10-bit I<sup>2</sup>C write transaction, per the NXP I<sup>2</sup>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 845x device tries to access the bus at the same time as another I<sup>2</sup>C master device and loses arbitration, the write transaction is terminated and an error is returned. If the slave device does not acknowledge the address, the NI 845x device ACK polls as specified in **i2c configuration in**. (Refer to the [I<sup>2</sup>C ACK Polling](#) section in Chapter 1, [Introduction](#), for more information about ACK polling.) If the slave does not acknowledge any transaction byte, an error is returned. Otherwise, the transaction is completed, and a stop condition is generated per the NXP I<sup>2</sup>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.

## NI-845x I<sup>2</sup>C Write Read.vi

### Purpose

Performs a write followed by read (combined format) on an I<sup>2</sup>C slave device.



### Inputs



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



**i2c configuration in** is a reference to a specific I<sup>2</sup>C 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<sup>2</sup>C slave.



**num bytes to read** specifies the number of bytes to read from the I<sup>2</sup>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



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



**i2c configuration out** is a reference to the I<sup>2</sup>C configuration after this VI runs.



**read data** contains an array of read data from the I<sup>2</sup>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 I<sup>2</sup>C Write Read.vi** to perform a write followed by read (combined format) on an I<sup>2</sup>C slave device. During the read portion of the transaction, per the NXP I<sup>2</sup>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<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7 or 10-bit I<sup>2</sup>C write/read transaction. Per the NXP I<sup>2</sup>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 845x device tries to access the bus at the same time as another I<sup>2</sup>C master device and loses arbitration, the read transaction is terminated and an error is returned. If the slave device does not acknowledge the address, the NI 845x device ACK polls as specified in **i2c configuration in**. (Refer to the [I<sup>2</sup>C ACK Polling](#) section in Chapter 1, [Introduction](#), for more information about ACK polling.) If the read address or a byte write within the transaction is not acknowledged, an error is returned. Otherwise, the transaction is completed and a stop condition is generated per the NXP I<sup>2</sup>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<sup>2</sup>C Specification provides flexibility in the construction of I<sup>2</sup>C transactions. The NI-845x I<sup>2</sup>C scripting VIs allow creating and customizing complex I<sup>2</sup>C transactions as needed.

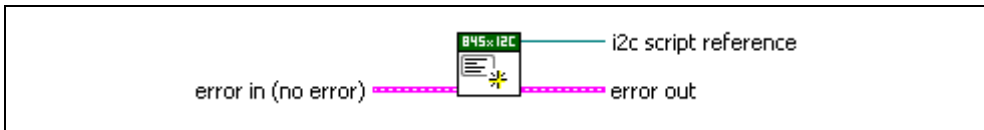
Before using **NI-845x I<sup>2</sup>C Write Read.vi**, you need to ensure that the configuration parameters specified in **i2c configuration in** are correct for the device you want to access.

# Scripting

## NI-845x I2C Create Script Reference.vi

### Purpose

Creates a new NI-845x I<sup>2</sup>C 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-845x I<sup>2</sup>C 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.

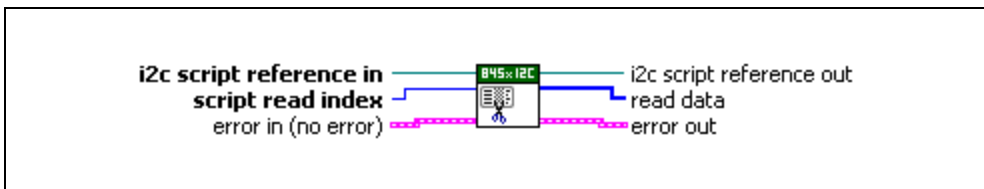
## Description

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

## NI-845x I<sup>2</sup>C Extract Script Read Data.vi

### Purpose

Extracts the desired read data from an I<sup>2</sup>C script, referenced by **i2c script reference in**, which has been processed by **NI-845x I<sup>2</sup>C Run Script.vi**. Each script read command (**NI-845x I<sup>2</sup>C Script Read.vi**, **NI-845x I<sup>2</sup>C Script DIO Read Port.vi**, **NI-845x I<sup>2</sup>C 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 I<sup>2</sup>C Extract Script Read Data.vi**.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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

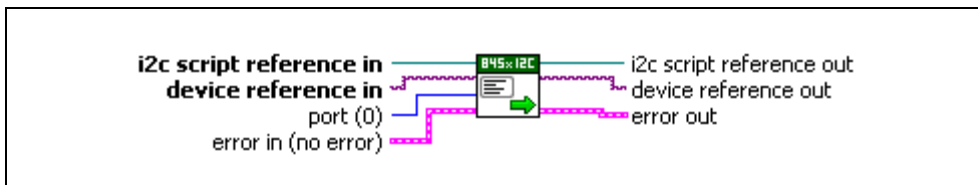
Use **NI-845x I2C Extract Script Read Data.vi** to extract the desired read data from an I<sup>2</sup>C script, referenced by **i2c script reference in**, which has been processed by **NI-845x I2C Run Script.vi**. Each I<sup>2</sup>C script read command (**NI-845x I2C Script Read.vi**, **NI-845x I2C 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 I<sup>2</sup>C Run Script.vi

### Purpose

Executes an I<sup>2</sup>C 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 I<sup>2</sup>C script that is run on an NI 845x device.



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



**port** specifies the I<sup>2</sup>C 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<sup>2</sup>C script after this VI runs.



**device reference out** is a reference to the NI 845x 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.

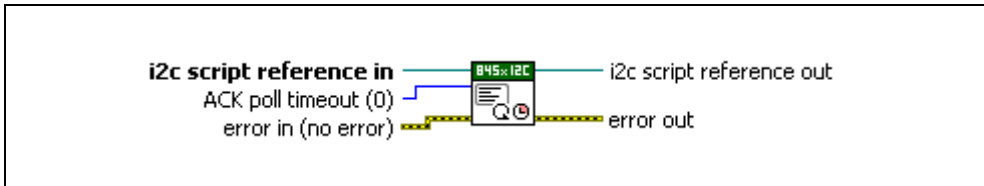
## Description

Use **NI-845x I2C Run Script.vi** to execute an I<sup>2</sup>C script referenced by **i2c script reference in** on the device referenced by **device reference in**. You must first create an I<sup>2</sup>C script using the I<sup>2</sup>C scripting VIs. Next, you wire its script reference into **i2c script reference in**. If you have multiple NI 845x devices installed in your system, you can select which device to write your I<sup>2</sup>C script to by wiring its device reference to **device reference in**. If your NI 845x device supports multiple I<sup>2</sup>C ports, you can also select which port to write your I<sup>2</sup>C script to. For single I<sup>2</sup>C port NI 845x devices, you must use the default port (0). In this way, you can create one script to run on various NI 845x devices, on various I<sup>2</sup>C ports within those devices. **NI-845x I2C Run Script.vi** loads and executes your I<sup>2</sup>C script on the NI 845x device and I<sup>2</sup>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 I<sup>2</sup>C Script ACK Poll Timeout.vi

## Purpose

Adds an I<sup>2</sup>C Script ACK Poll Timeout command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C ACK poll timeout.



## Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script run on an NI 845x device.



**ACK poll timeout** specifies the timeout for ACK poll in milliseconds.



**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<sup>2</sup>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.

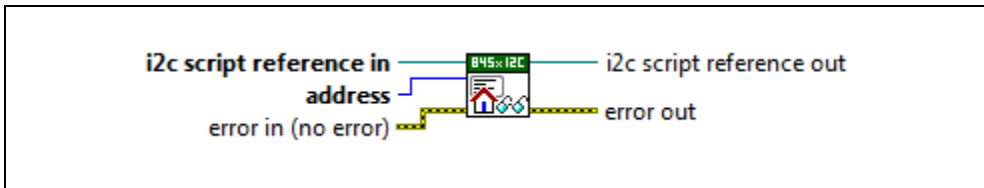
## Description

Use **NI-845x I2C ACK Poll Timeout.vi** to add an I<sup>2</sup>C ACK Poll Timeout command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C ACK poll timeout when you use **NI-845x I2C Run Script.vi** to execute the script. When **ACK poll timeout** is zero, ACK polling is disabled. A nonzero **ACK poll timeout** causes **NI-845x I2C Script Address+Write.vi** and **NI-845x I2C Script Address+Read.vi** to ACK poll until an acknowledge (ACK) is detected or the timeout is reached. If enabled, ACK polling occurs after either a start condition or a restart condition. **ACK poll timeout** persists throughout the script and is reset to the default (0) at the end of the script.

## NI-845x I<sup>2</sup>C Script Address+Read.vi

### Purpose

Adds an I<sup>2</sup>C Script Address+Read command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I<sup>2</sup>C bus. The direction bit is internally set to 1 for read.



### Inputs



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



**address** specifies the 7-bit address to read. For 7-bit device addressing, the NXP I<sup>2</sup>C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845x 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-845x 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<sup>2</sup>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.

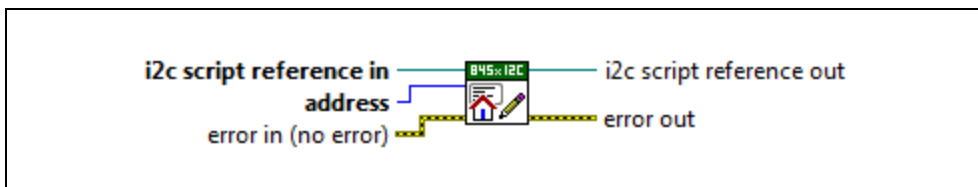
## Description

Use **NI-845x I<sup>2</sup>C Script Address+Read.vi** to add an I<sup>2</sup>C Script Address+Read command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you use **NI-845x I<sup>2</sup>C 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<sup>2</sup>C bus using an I<sup>2</sup>C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I<sup>2</sup>C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, the NI 845x device ACK polls as set in **NI-845x I<sup>2</sup>C Script ACK Poll Timeout.vi**. (Refer to the *I<sup>2</sup>C ACK Polling* section in Chapter 1, *Introduction*, for more information about ACK polling.)

## NI-845x I<sup>2</sup>C Script Address+Write.vi

### Purpose

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



### Inputs



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



**address** specifies the 7-bit address to write. For 7-bit device addressing, the NXP I<sup>2</sup>C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C Specification and defines an address for a 7-bit device as a 7-bit value. The NI-845x 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-845x 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<sup>2</sup>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.

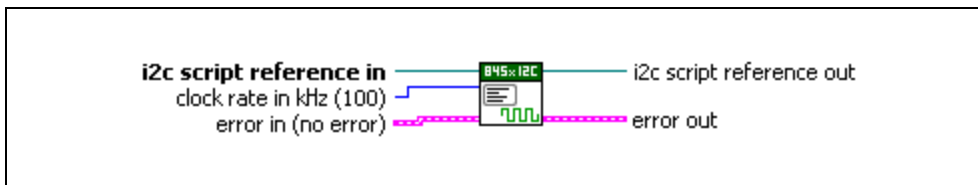
## Description

Use **NI-845x I<sup>2</sup>C Script Address+Write.vi** to add an I<sup>2</sup>C Script Address+Write command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes a 7-bit address to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you use **NI-845x I<sup>2</sup>C 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<sup>2</sup>C bus using an I<sup>2</sup>C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I<sup>2</sup>C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, the NI 845x device ACK polls as set in **NI-845x I<sup>2</sup>C Script ACK Poll Timeout.vi**. (Refer to the *I<sup>2</sup>C ACK Polling* section in Chapter 1, *Introduction*, for more information about ACK polling.)

## NI-845x I<sup>2</sup>C Script Clock Rate.vi

### Purpose

Adds an I<sup>2</sup>C Script Clock Rate command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C clock rate.



### Inputs



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



**clock rate in kHz** specifies the I<sup>2</sup>C clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845x 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<sup>2</sup>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.

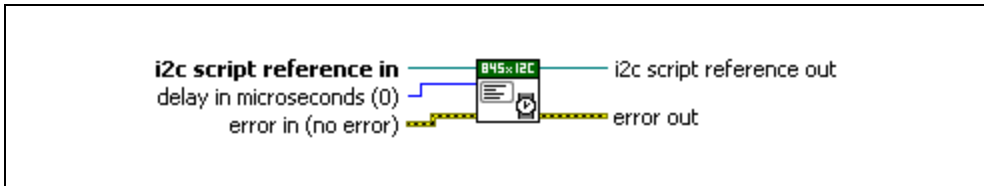
## Description

Use **NI-845x I2C Script Clock Rate.vi** to add an I<sup>2</sup>C Script Clock Rate command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C clock rate for the I<sup>2</sup>C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The NI 845x 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.

## NI-845x I<sup>2</sup>C Script Delay (Microsecond).vi

### Purpose

Adds an I<sup>2</sup>C Script Delay command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command adds a microsecond delay after the previous I<sup>2</sup>C script command.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script run on an NI 845x device.



**delay in microseconds** 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<sup>2</sup>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.

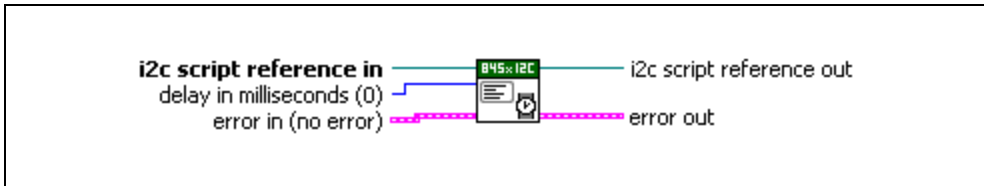
## Description

Use **NI-845x I<sup>2</sup>C Script Delay.vi** to add an I<sup>2</sup>C Script  $\mu$ s Delay command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command adds a microsecond delay after the previous I<sup>2</sup>C script command.

## NI-845x I<sup>2</sup>C Script Delay (Millisecond).vi

### Purpose

Adds an I<sup>2</sup>C Script Delay command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command adds a millisecond delay after the previous I<sup>2</sup>C script command.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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.

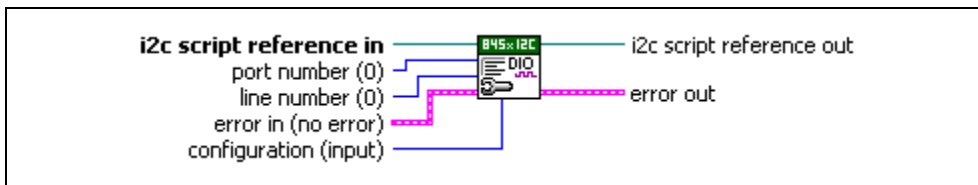
## Description

Use **NI-845x I<sup>2</sup>C Script Delay.vi** to add an I<sup>2</sup>C Script ms Delay command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command adds a millisecond delay after the previous I<sup>2</sup>C script command.

## NI-845x I<sup>2</sup>C Script DIO Configure Line.vi

### Purpose

Adds an I<sup>2</sup>C Script DIO Configure Line command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command configures a DIO line on an NI 845x device.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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 I<sup>2</sup>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.

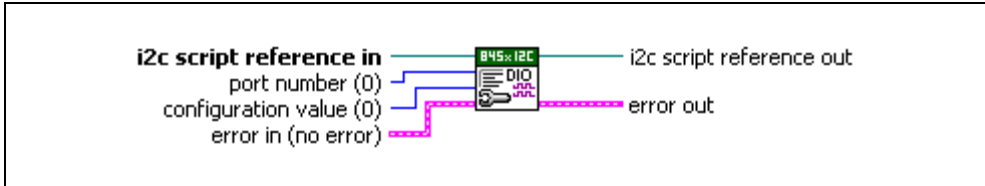
## Description

Use **NI-845x I2C Script DIO Configure Line.vi** to add an I<sup>2</sup>C Script DIO Configure Line command to an I<sup>2</sup>C 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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x devices with one DIO port, **port number** must be left at the default (0).

# NI-845x I<sup>2</sup>C Script DIO Configure Port.vi

## Purpose

Adds an I<sup>2</sup>C Script DIO Configure Port command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command configures a DIO port on an NI 845x device.



## Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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.

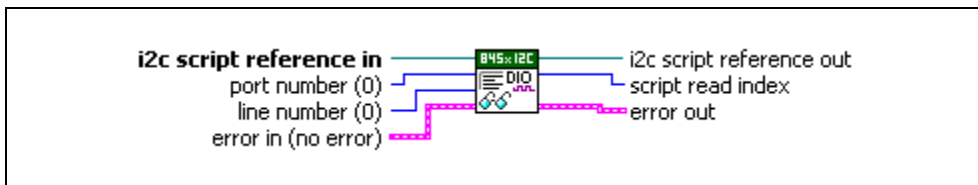
## Description

Use **NI-845x I2C Script DIO Configure Port.vi** to add an I<sup>2</sup>C Script DIO Configure Port command to an I<sup>2</sup>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 845x devices with multiple DIO ports, use the **port number** input to select the port to configure. For NI 845x devices with one DIO port, **port number** must be left at the default (0).

## NI-845x I<sup>2</sup>C Script DIO Read Line.vi

### Purpose

Adds an I<sup>2</sup>C Script DIO Read Line command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command reads from a DIO line on an NI 845x device.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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

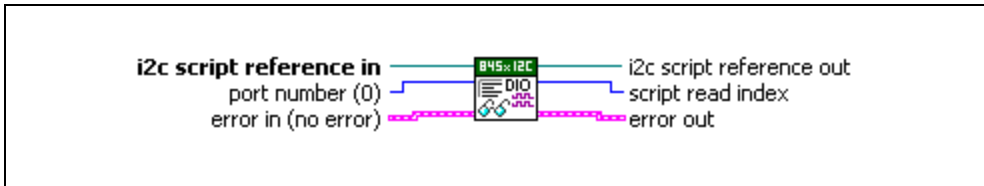
Use **NI-845x I2C Script DIO Read Line.vi** to add an I<sup>2</sup>C Script DIO Read command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command allows you to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 I<sup>2</sup>C Script DIO Read Port.vi

### Purpose

Adds an I<sup>2</sup>C Script DIO Read Port command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command reads from a DIO port on an NI 845x device.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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 I<sup>2</sup>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 I<sup>2</sup>C 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

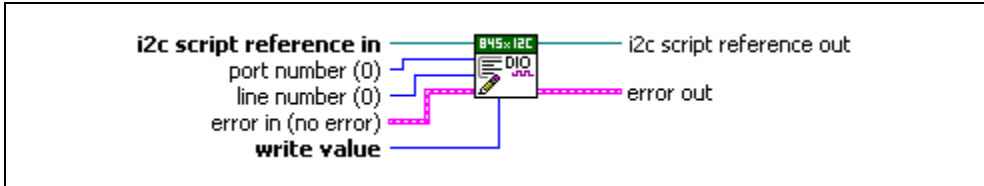
Use **NI-845x I<sup>2</sup>C Script DIO Read Port.vi** to add an I<sup>2</sup>C Script DIO Read Port command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command allows you to read all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 I<sup>2</sup>C Extract Script Read Data.vi** after script execution, which returns the data byte read by this script command.

# NI-845x I<sup>2</sup>C Script DIO Write Line.vi

## Purpose

Adds an I<sup>2</sup>C Script DIO Write Line command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes to a DIO line on an NI 845x device.



## Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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.

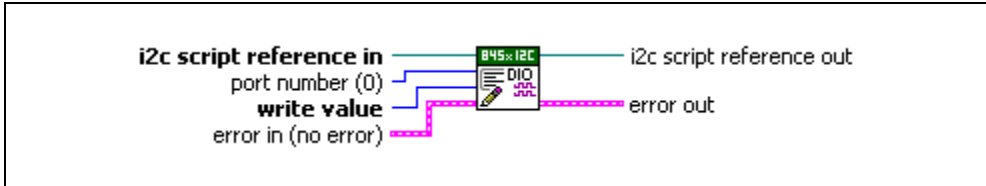
## Description

Use **NI-845x I2C Script DIO Write Line.vi** to add an I<sup>2</sup>C Script DIO Write Line command to an I<sup>2</sup>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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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<sup>2</sup>C Script DIO Write Port command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes to a DIO port on an NI 845x device.



## Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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 I<sup>2</sup>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.

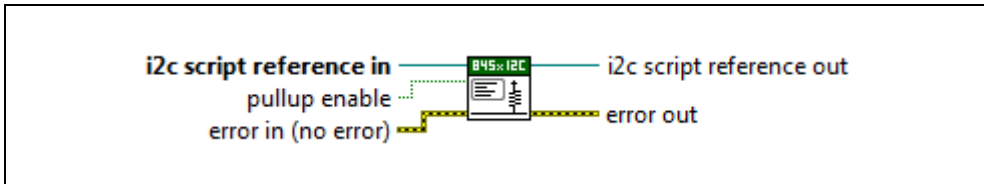
## Description

Use **NI-845x I2C Script DIO Write Port.vi** to add an I<sup>2</sup>C Script DIO Write Port command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command allows you to write all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x devices with one DIO port, **port number** must be left at the default (0).

## NI-845x I<sup>2</sup>C Script Pullup Enable.vi

### Purpose

Adds an I<sup>2</sup>C Script Pullup Enable command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command enables or disables the internal pullups on an NI 845x device.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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 I<sup>2</sup>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.

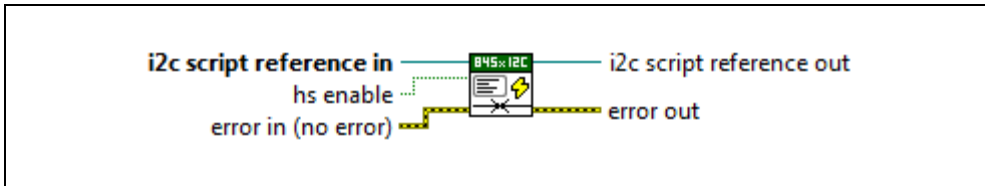
## Description

Use **NI-845x I<sup>2</sup>C Script Pullup Enable.vi** to add an I<sup>2</sup>C Script Pullup Enable command to an I<sup>2</sup>C script referenced by **i2c script reference in**. Use this command to set the status of onboard pullups for I<sup>2</sup>C operations. The pullup resistors pull SDA and SCL up to [I/O Voltage Level](#).

# NI-845x I<sup>2</sup>C Script HS Enable.vi

## Purpose

Adds an I<sup>2</sup>C Script HS Enable command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command enables or disables High Speed mode on an NI 845x device.



## Inputs



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



**hs enable** sets the High Speed mode to enabled or disabled on an NI 845x 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<sup>2</sup>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

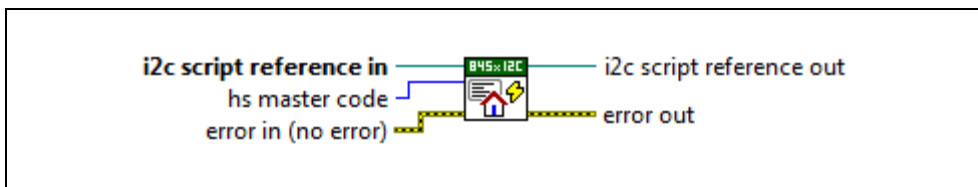
Use **NI 845x I<sup>2</sup>C Script HS Enable.vi** to add an I<sup>2</sup>C Script HS Enable command to an I<sup>2</sup>C script referenced by **i2c script reference in**. Use this command to enable High Speed mode. High Speed mode must be enabled to use the High Speed clock rate or the High Speed master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

High Speed mode is described in the NXP I<sup>2</sup>C Specification.

## NI-845x I<sup>2</sup>C Script HS Master Code.vi

### Purpose

Adds an I<sup>2</sup>C Script HS Master Code command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command transfers the master code for High Speed mode on an NI 845x device.



### Inputs



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



**hs master code** sets the lower 3 bits of the master code on the NI 845x 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<sup>2</sup>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.

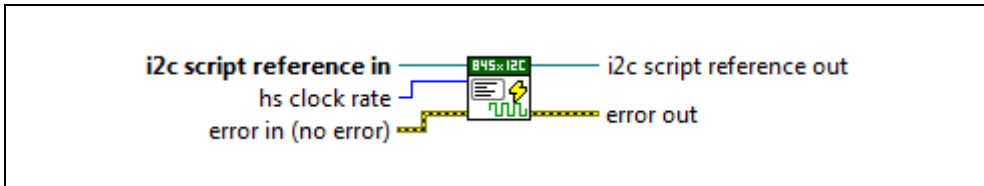
## Description

Use **NI 845x I2C HS Master Code.vi** to add an I<sup>2</sup>C Script HS Master Code command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes a master code to the I<sup>2</sup>C bus connected to the I<sup>2</sup>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<sup>2</sup>C bus using an I<sup>2</sup>C script start command. The master code is internally set to 00001XXX. The lower three bits are set using the I<sup>2</sup>C Script HS Master Code command. After the master code is transferred, the device waits for the slave device on the I<sup>2</sup>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 I<sup>2</sup>C Script HS Clock Rate.vi

### Purpose

Adds an I<sup>2</sup>C Script HS Clock Rate command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C High Speed clock rate.



### Inputs



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



**hs clock rate** specifies the I<sup>2</sup>C High Speed clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which clock rates your NI 845x 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<sup>2</sup>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.

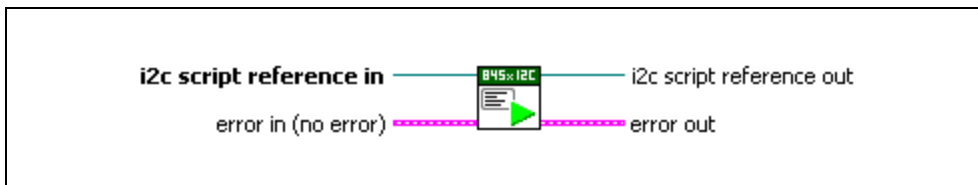
## Description

Use **NI-845x I2C Script HS Clock Rate.vi** to add an I<sup>2</sup>C Script High Speed Clock Rate command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command sets the I<sup>2</sup>C High Speed clock rate for the I<sup>2</sup>C port you specify when you use **NI-845x I2C Run Script.vi** to execute the script. The NI 845x 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.

## NI-845x I<sup>2</sup>C Script Issue Start.vi

### Purpose

Adds an I<sup>2</sup>C Script Issue Start command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command issues a start condition on the I<sup>2</sup>C bus.



### Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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.

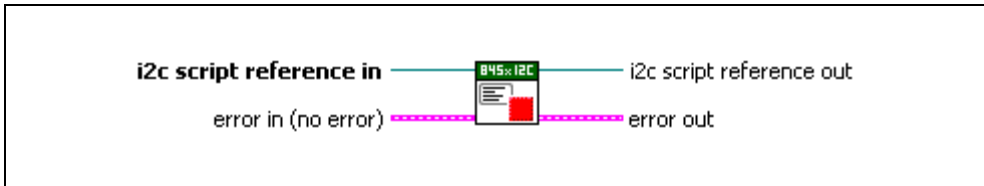
## Description

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

# NI-845x I<sup>2</sup>C Script Issue Stop.vi

## Purpose

Adds an I<sup>2</sup>C Script Issue Stop command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command issues a stop condition on the I<sup>2</sup>C bus.



## Inputs



**i2c script reference in** is a reference to an I<sup>2</sup>C script that is run on an NI 845x 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<sup>2</sup>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.



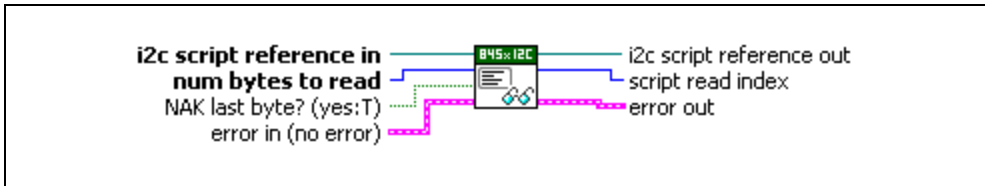
## Description

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

# NI-845x I2C Script Read.vi

## Purpose

Adds an I<sup>2</sup>C Script Read command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command reads an array of data from an I<sup>2</sup>C slave device.



## Inputs



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



**num bytes to read** specifies the number of bytes to read from an I<sup>2</sup>C slave.



**NAK Last Byte?** sets whether the last byte read is acknowledged (FALSE) or not acknowledged (TRUE) by the I<sup>2</sup>C 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<sup>2</sup>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 I<sup>2</sup>C 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

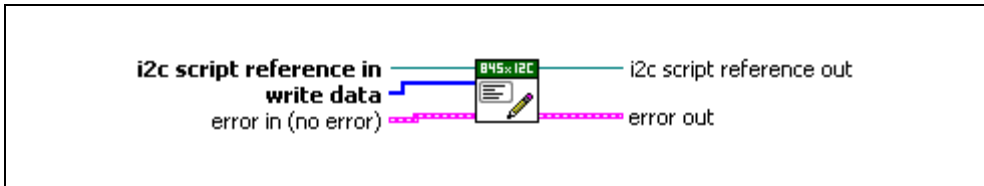
Use **NI-845x I<sup>2</sup>C Script Read.vi** to add an I<sup>2</sup>C Script Read command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command reads an array of data from a device connected to the I<sup>2</sup>C port you specify when you use **NI-845x I<sup>2</sup>C Run Script.vi** to execute the script. This command assumes that a start condition and address+read condition have been issued to the I<sup>2</sup>C bus using prior I<sup>2</sup>C script commands. It clocks in **num bytes to read** bytes from the I<sup>2</sup>C slave device, acknowledging each byte up to the last one. Depending on the type of I<sup>2</sup>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<sup>2</sup>C port, you can wire **script read index** to **NI-845x I<sup>2</sup>C Extract Script Read Data.vi** after execution of the script, which returns the data read by this script command.

## NI-845x I<sup>2</sup>C Script Write.vi

### Purpose

Adds an I<sup>2</sup>C Script Write command to an I<sup>2</sup>C script referenced by **i2c script reference in**. This command writes an array of data to an I<sup>2</sup>C slave device.



### Inputs



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



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

---

# NI-845x I<sup>2</sup>C API for C

This chapter lists the functions for the NI-845x I<sup>2</sup>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 scripting.

## Section Headings

---

The NI-845x I<sup>2</sup>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-845x I<sup>2</sup>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')
NiHandle	Operating system independent handle

## List of Functions

The following table contains an alphabetical list of the NI-845x I<sup>2</sup>C API for C functions.

Function	Purpose
<a href="#">ni845xClose</a>	Closes a previously opened NI 845x device.
<a href="#">ni845xCloseFindDeviceHandle</a>	Closes the handles created by <a href="#">ni845xFindDevice</a> .
<a href="#">ni845xDeviceLock</a>	Locks NI 845x devices for access by a single thread.
<a href="#">ni845xDeviceUnlock</a>	Unlocks NI 845x devices.
<a href="#">ni845xFindDevice</a>	Finds an NI 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using <a href="#">ni845xFindDeviceNext</a> .
<a href="#">ni845xFindDeviceNext</a>	Finds subsequent devices after <a href="#">ni845xFindDevice</a> has been called.
<a href="#">ni845xI2cConfigurationClose</a>	Closes an NI-845x I <sup>2</sup> C I/O configuration.

Function	Purpose
<a href="#">ni845xI2cConfigurationGetAckPollTimeout</a>	Retrieves the configuration's ACK poll timeout in milliseconds.
<a href="#">ni845xI2cConfigurationGetAddress</a>	Retrieves the configuration's address.
<a href="#">ni845xI2cConfigurationGetAddressSize</a>	Retrieves the configuration's address size.
<a href="#">ni845xI2cConfigurationGetClockRate</a>	Retrieves the configuration's clock rate in kilohertz.
<a href="#">ni845xI2cConfigurationGetHSClockRate</a>	Retrieves the configuration's High Speed clock rate in kilohertz.
<a href="#">ni845xI2cConfigurationGetHSEnable</a>	Retrieves the configuration's High Speed enable setting.
<a href="#">ni845xI2cConfigurationGetHSMasterCode</a>	Retrieves the configuration's High Speed master code.
<a href="#">ni845xI2cConfigurationGetPort</a>	Retrieves the configuration's port value.
<a href="#">ni845xI2cConfigurationOpen</a>	Creates a new NI-845x I <sup>2</sup> C configuration.
<a href="#">ni845xI2cConfigurationSetAckPollTimeout</a>	Sets the configuration's ACK poll timeout in milliseconds.
<a href="#">ni845xI2cConfigurationSetAddress</a>	Sets the configuration's address.
<a href="#">ni845xI2cConfigurationSetAddressSize</a>	Sets the configuration's address size.
<a href="#">ni845xI2cConfigurationSetClockRate</a>	Sets the configuration's clock rate in kilohertz.
<a href="#">ni845xI2cConfigurationSetHSClockRate</a>	Sets the configuration's High Speed clock rate in kilohertz.
<a href="#">ni845xI2cConfigurationSetHSEnable</a>	Sets the configuration's High Speed enable setting.
<a href="#">ni845xI2cConfigurationSetHSMasterCode</a>	Sets the configuration's High Speed master code.
<a href="#">ni845xI2cConfigurationSetPort</a>	Sets the configuration's port number.
<a href="#">ni845xI2cRead</a>	Reads an array of data from an I <sup>2</sup> C slave device.



Function	Purpose
<code>ni845xI2cScriptAckPollTimeout</code>	Adds an I <sup>2</sup> C Script ACK Poll Timeout command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command sets the I <sup>2</sup> C ACK poll timeout.
<code>ni845xI2cScriptAddressRead</code>	Adds an I <sup>2</sup> C Script Address+Read command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command writes a 7-bit address to the I <sup>2</sup> C bus. The direction bit is internally set to 1 for read. If the ACK poll timeout is nonzero, this command performs ACK polling.
<code>ni845xI2cScriptAddressWrite</code>	Adds an I <sup>2</sup> C Script Address+Write command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command writes a 7-bit address to the I <sup>2</sup> C bus. The direction bit is internally set to 0 for write. If the ACK poll timeout is nonzero, this command performs ACK polling.
<code>ni845xI2cScriptClockRate</code>	Adds an I <sup>2</sup> C Script Clock Rate command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command sets the I <sup>2</sup> C clock rate.
<code>ni845xI2cScriptClose</code>	Closes an I <sup>2</sup> C script.
<code>ni845xI2cScriptDelay</code>	Adds an I <sup>2</sup> C Script Delay command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command adds a millisecond delay after the previous I <sup>2</sup> C script command.
<code>ni845xI2cScriptDioConfigureLine</code>	Adds an I <sup>2</sup> C Script DIO Configure Line command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command configures a DIO line on an NI 845x device.
<code>ni845xI2cScriptDioConfigurePort</code>	Adds an I <sup>2</sup> C Script DIO Configure Port command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command configures a DIO port on an NI 845x device.

Function	Purpose
<a href="#">ni845xI2cScriptDioReadLine</a>	Adds an I <sup>2</sup> C Script DIO Read Line command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command reads from a DIO line on an NI 845x device.
<a href="#">ni845xI2cScriptDioReadPort</a>	Adds an I <sup>2</sup> C Script DIO Read Port command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command reads from a DIO port on an NI 845x device.
<a href="#">ni845xI2cScriptDioWriteLine</a>	Adds an I <sup>2</sup> C Script DIO Write Line command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command writes to a DIO line on an NI 845x device.
<a href="#">ni845xI2cScriptDioWritePort</a>	Adds an I <sup>2</sup> C Script DIO Write Port command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command writes to a DIO port on an NI 845x device.
<a href="#">ni845xI2cScriptPullupEnable</a>	Adds an I <sup>2</sup> C Script Pullup Enable command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command enables or disables the internal I <sup>2</sup> C pullup resistors. The pullups connect to <a href="#">ni845xSetIoVoltageLevel</a> .
<a href="#">ni845xI2cScriptExtractReadData</a>	Extracts the desired read data from an I <sup>2</sup> C script, referenced by <code>ScriptHandle</code> , which has been processed by <a href="#">ni845xI2cScriptRun</a> . Each script read command ( <a href="#">ni845xI2cScriptRead</a> , <a href="#">ni845xI2cScriptDioReadPort</a> , <a href="#">ni845xI2cScriptDioReadLine</a> ) returns a script read index. You can extract data for each script read index in a script, by passing each index to <a href="#">ni845xI2cScriptExtractReadData</a> .

Function	Purpose
<code>ni845xI2cScriptExtractReadDataSize</code>	Retrieves the read data size from an I <sup>2</sup> C script, referenced by <code>ScriptHandle</code> , which has been processed by <code>ni845xI2cScriptRun</code> . Each script read command ( <code>ni845xI2cScriptRead</code> , <code>ni845xI2cScriptDioReadPort</code> , <code>ni845xI2cScriptDioReadLine</code> ) returns a script read index. You can extract data for each script read index in a script, by passing each index to <code>ni845xI2cScriptExtractReadData</code> .
<code>ni845xI2cScriptHSEnable</code>	Adds an I <sup>2</sup> C Script HS Enable command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command enables the I <sup>2</sup> C port to run in high-speed mode.
<code>ni845xI2cScriptHSMasterCode</code>	Adds an I <sup>2</sup> C Script HS Master Code command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command configures the I <sup>2</sup> C master code, which is used to initiate High Speed I <sup>2</sup> C mode.
<code>ni845xI2cScriptHSClockRate</code>	Adds an I <sup>2</sup> C Script HS Clock Rate command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command sets the High Speed I <sup>2</sup> C clock rate.
<code>ni845xI2cScriptIssueStart</code>	Adds an I <sup>2</sup> C Script Issue Start command to an I <sup>2</sup> C script indicated by <code>ScriptHandle</code> . This command issues a start condition on the I <sup>2</sup> C bus.
<code>ni845xI2cScriptIssueStop</code>	Adds an I <sup>2</sup> C Script Issue Stop command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command issues a stop condition on the I <sup>2</sup> C bus.
<code>ni845xI2cScriptOpen</code>	Opens an empty I <sup>2</sup> C script to begin adding commands to.

Function	Purpose
<code>ni845xI2cScriptRead</code>	Adds an I <sup>2</sup> C Script Read command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command reads an array of data from an I <sup>2</sup> C slave device.
<code>ni845xI2cScriptReset</code>	Resets an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> to an empty state.
<code>ni845xI2cScriptRun</code>	Sends the I <sup>2</sup> C script to the desired NI 845x device, which then interprets and runs it.
<code>ni845xI2cScriptUsDelay</code>	Adds an I <sup>2</sup> C Script $\mu$ s Delay command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command adds a microsecond delay after the previous I <sup>2</sup> C script command.
<code>ni845xI2cScriptWrite</code>	Adds an I <sup>2</sup> C Script Write command to an I <sup>2</sup> C script referenced by <code>ScriptHandle</code> . This command writes an array of data to an I <sup>2</sup> C slave device.
<code>ni845xI2cSetPullupEnable</code>	Enables or disables the onboard I <sup>2</sup> C pullups.
<code>ni845xI2cWrite</code>	Writes an array of data to an I <sup>2</sup> C slave device.
<code>ni845xI2cWriteRead</code>	Performs a write followed by read (combined format) on an I <sup>2</sup> C slave device.
<code>ni845xOpen</code>	Opens an NI 845x device for use with various write, read, and device property functions.
<code>ni845xSetIoVoltageLevel</code>	Sets the voltage level of the NI-845x I/O pins (DIO/SPI/VioRef).
<code>ni845xSetTimeout</code>	Sets the global timeout value.
<code>ni845xStatusToString</code>	Converts a status code into a descriptive string.

# General Device

---

## ni845xClose

---

### Purpose

Closes a previously opened NI 845x device.

### Format

```
int32 ni845xClose(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 (  
    NiHandle FindDeviceHandle  
);
```

### Inputs

NiHandle 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 845x devices for access by a single thread.

### Format

```
int32 ni845xDeviceLock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x devices.

### Format

```
int32 ni845xDeviceUnlock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using [ni845xFindDeviceNext](#).

### Format

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

### Inputs

None.

### Outputs

`char * pFirstDevice`

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

`NiHandle * 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 845x 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 845x device and the number of NI 845x 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 845x 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 (
    NiHandle FindDeviceHandle,
    char * pNextDevice
);
```

### Inputs

NiHandle FindDeviceHandle

Describes a find list. [ni845xFindDevice](#) creates this parameter.

### Outputs

char \* pNextDevice

A pointer to the string containing the next NI 845x 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 845x device for use with various write, read, and device property functions.

### Format

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

### Inputs

char \* pResourceName

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

### Outputs

NiHandle \* 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 845x 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 845x device.

### Format

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

### Inputs

NiHandle 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 845x device. The board reference voltage is used for SPI, I<sup>2</sup>C, 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 845x device.

### Format

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

### Inputs

NiHandle 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 I<sup>2</sup>C 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-845x 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-845x 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-845x function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845x use of this status.

## NI-845x 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-845x function. If an error is detected, you should close all NI-845x 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<sup>2</sup>C 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.



## ni845xSetTimeout

---

### Purpose

Modifies the global timeout for operations when using an NI 845x device.

### Format

```
int32 ni845xSetTimeout (
    NiHandle DeviceHandle,
    uInt32 Timeout
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

uInt32 Timeout

The timeout value in milliseconds. The minimum timeout is 1000 ms (1 second).

### 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 `ni845xI2cSetTimeout` to set the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.

The default of this property is 30000 (30 seconds).



**Note** You should set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

# Configuration

---

## ni845xI2cConfigurationClose

---

### Purpose

Closes an I<sup>2</sup>C I/O configuration.

### Format

```
int32 ni845xI2cConfigurationClose (  
    NiHandle ConfigurationHandle  
);
```

### Inputs

NiHandle 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.

## ni845xI2cConfigurationGetAckPollTimeout

---

### Purpose

Retrieves the configuration ACK poll timeout in milliseconds.

### Format

```
int32 ni845xI2cConfigurationGetAckPollTimeout (
    NiHandle ConfigurationHandle,
    uInt16 * pTimeout
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

### Outputs

uInt16 \* pTimeout

A pointer to an unsigned 16-bit integer to store the ACK poll timeout 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 [ni845xI2cConfigurationGetAckPollTimeout](#) to retrieve the I<sup>2</sup>C ACK poll timeout in milliseconds.

## ni845xI2cConfigurationGetAddress

---

### Purpose

Retrieves the configuration address.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

### Outputs

uInt16 \* pAddress

A pointer to an unsigned 16-bit integer to store the I<sup>2</sup>C 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<sup>2</sup>C configuration slave address without the direction bit.

## ni845xI2cConfigurationGetAddressSize

---

### Purpose

Retrieves the configuration address size.

### Format

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

### Inputs

NiHandle 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<sup>2</sup>C slave device this configuration describes.

## ni845xI2cConfigurationGetClockRate

---

### Purpose

Retrieves the configuration clock rate in kilohertz.

### Format

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

### Inputs

NiHandle 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<sup>2</sup>C clock rate in kilohertz. This retrieves the value currently stored in memory, which may not be compatible with your NI 845x device.

## ni845xI2cConfigurationGetHSClockRate

---

### Purpose

Retrieves the configuration High Speed clock rate in kilohertz.

### Format

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

### Inputs

NiHandle 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 I<sup>2</sup>C High Speed clock rate in kilohertz. This retrieves the value currently stored in memory, which may not be compatible with your NI 845x device.

## ni845xI2cConfigurationGetHSEnable

---

### Purpose

Retrieves the configuration High Speed enable status.

### Format

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

### Inputs

NiHandle 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 845x device.



## ni845xI2cConfigurationGetHSMasterCode

---

### Purpose

Retrieves the configuration master code.

### Format

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

### Inputs

NiHandle 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<sup>2</sup>C High Speed master code. This retrieves the value currently stored in memory, which may not be compatible with your NI 845x device.

## ni845xI2cConfigurationGetPort

---

### Purpose

Retrieves the configuration port value.

### Format

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

### Inputs

NiHandle 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<sup>2</sup>C port that this configuration communicates across.

## ni845xI2cConfigurationOpen

---

### Purpose

Creates a new NI-845x I<sup>2</sup>C configuration.

### Format

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

### Inputs

None.

### Outputs

NiHandle \* 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-845x I<sup>2</sup>C Basic API. Pass the handles to the `ni845xI2cConfigurationSet*` series of functions to modify the configuration properties. Then, pass the configuration to the I<sup>2</sup>C basic functions to execute them on the described I<sup>2</sup>C slave. After you finish communicating with your I<sup>2</sup>C slave, pass the handle to the `ni845xI2cConfigurationSet*` series of functions to reconfigure it or use [ni845xI2cConfigurationClose](#) to delete the configuration.

## ni845xI2cConfigurationSetAckPollTimeout

---

### Purpose

Sets the configuration ACK poll timeout in milliseconds.

### Format

```
int32 ni845xI2cConfigurationSetAckPollTimeout (
    NiHandle ConfigurationHandle,
    uInt16 Timeout
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt16 Timeout

Specifies the I<sup>2</sup>C ACK poll timeout in milliseconds. When this value is zero, ACK polling is disabled. Otherwise, the [ni845xI2cRead](#), [ni845xI2cWrite](#), and [ni845xI2cWriteRead](#) API calls ACK poll until an acknowledge (ACK) is detected or the timeout is reached.

ACK polling is not supported with 10-bit addressing. If the configuration's address size is set to 10 bits and Timeout is nonzero, an error is generated when attempting an I/O API call.

The default value is 0 ms (disabled).

### 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 [ni845xI2cConfigurationSetAckPollTimeout](#) to set the I<sup>2</sup>C configuration ACK Poll Timeout in milliseconds.

## ni845xI2cConfigurationSetAddress

---

### Purpose

Sets the configuration address.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt16 Address

The slave address. For 7-bit device addressing, the NXP I<sup>2</sup>C specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C specification and defines an address for a 7-bit device as a 7-bit value. The NI-845x 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-845x 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 I<sup>2</sup>C slave address. Do not include the direction bit.

## ni845xI2cConfigurationSetAddressSize

---

### Purpose

Sets the configuration address size.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

int32 Size

The addressing scheme to use when addressing the I<sup>2</sup>C slave device this configuration describes. Size uses the following values:

- `kNi845xI2cAddress7Bit (0)`: The NI 845x hardware uses the standard 7-bit addressing when communicating with the I<sup>2</sup>C slave device.
- `kNi845xI2cAddress10Bit (1)`: The NI 845x hardware uses the extended 10-bit addressing when communicating with the I<sup>2</sup>C 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 (  
    NiHandle ConfigurationHandle,  
    uInt16 ClockRate  
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt16 ClockRate

Specifies the I<sup>2</sup>C clock rate in kilohertz. Refer to Chapter 3, [NI USB-845x Hardware Overview](#), to determine which clock rates your NI 845x 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 I<sup>2</sup>C configuration clock rate in kilohertz.

## ni845xI2cConfigurationSetHSClockRate

---

### Purpose

Sets the configuration High Speed clock rate in kilohertz.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt16 HSClockRate

Specifies the I<sup>2</sup>C clock rate in kilohertz. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), to determine which High Speed clock rates your NI 845x 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 I<sup>2</sup>C configuration High Speed clock rate in kilohertz.



## ni845xI2cConfigurationSetHSEnable

---

### Purpose

Sets the configuration High Speed enabled status.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt8 HSEnable

Specifies the I<sup>2</sup>C High Speed enabled status. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), to determine if your NI 845x device supports I<sup>2</sup>C High Speed mode. If your hardware does not support I<sup>2</sup>C 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 I<sup>2</sup>C High Speed enabled status.

## ni845xI2cConfigurationSetHSMasterCode

---

### Purpose

Sets the configuration High Speed master code.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt8 HSMasterCode

Specifies the I<sup>2</sup>C 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<sup>2</sup>C configuration High Speed master code.

## ni845xI2cConfigurationSetPort

---

### Purpose

Sets the configuration port number.

### Format

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

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cConfigurationOpen](#).

uInt8 Port

Specifies the I<sup>2</sup>C port that this configuration communicates across.

Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine the number of I<sup>2</sup>C ports your NI 845x 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<sup>2</sup>C slave device resides.

# Basic

---

## ni845xI2cRead

---

### Purpose

Reads an array of data from an I<sup>2</sup>C slave device.

### Format

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

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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<sup>2</sup>C slave device. Per the NXP I<sup>2</sup>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<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7-bit or 10-bit I<sup>2</sup>C read transaction, per the NXP I<sup>2</sup>C specification. The address type (7-bit or 10-bit) and other configuration parameters are specified by `ConfigurationHandle`. If the NI 845x device tries to access the bus at the same time as another I<sup>2</sup>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, the NI 845x device ACK polls as specified by `ConfigurationHandle`. (Refer to the *I<sup>2</sup>C ACK Polling* section in Chapter 1, *Introduction*, for more information about ACK polling.) Otherwise, the transaction is completed, and a stop condition is generated per the NXP I<sup>2</sup>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<sup>2</sup>C slave device.

### Format

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

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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<sup>2</sup>C slave device. This function first waits for the I<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7-bit or 10-bit I<sup>2</sup>C write transaction, per the NXP I<sup>2</sup>C specification. The address type (7-bit or 10-bit) and other configuration parameters are specified by `ConfigurationHandle`. If the NI 845x device tries to access the bus at the same time as another I<sup>2</sup>C master device and loses arbitration, the write transaction is terminated and an

error is returned. If the slave device does not acknowledge the address, the NI 845x device ACK polls as specified by `ConfigurationHandle`. (Refer to the *I<sup>2</sup>C ACK Polling* section in Chapter 1, *Introduction*, for more information about ACK polling.) 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 I<sup>2</sup>C specification.

## ni845xI2cWriteRead

---

### Purpose

Performs a write followed by read (combined format) on an I<sup>2</sup>C slave device.

### Format

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

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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 I<sup>2</sup>C slave device. During the transaction read portion, per the NXP I<sup>2</sup>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<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned. If the bus is free before the timeout, the NI 845x device executes a 7-bit or 10-bit I<sup>2</sup>C write/read transaction. Per the NXP I<sup>2</sup>C 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 845x device tries to access the bus at the same time as another I<sup>2</sup>C master device and loses arbitration, the read transaction is terminated and an error is returned. If the slave device does not acknowledge the write address, the NI 845x device ACK polls as specified by `ConfigurationHandle`. (Refer to the [I<sup>2</sup>C ACK Polling](#) section in Chapter 1, [Introduction](#), for more information about ACK polling.) If the slave device does not acknowledge the read 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 I<sup>2</sup>C specification. Note that this type of combined transaction is provided because it is commonly used (for example, with EEPROMs). The NXP I<sup>2</sup>C specification provides flexibility in the construction of I<sup>2</sup>C transactions. The NI-845x I<sup>2</sup>C scripting functions allow creating and customizing complex I<sup>2</sup>C 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.

# Scripting

---

## ni845xI2cScriptAckPollTimeout

---

### Purpose

Adds an I<sup>2</sup>C Script AckPoll Timeout command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C ACK poll timeout.

### Format

```
int32 ni845xI2cScriptAckPollTimeout (
    NiHandle ScriptHandle,
    uInt16   Timeout
);
```

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt16 Timeout`

The ACK poll timeout 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 `ni845xI2cScriptAckPollTimeout` to add an I<sup>2</sup>C Script ACK poll timeout command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C ACK poll timeout when you use [ni845xI2cScriptRun](#) to execute the script. When ACK poll timeout is zero, ACK polling is disabled. A nonzero ACK poll timeout causes [ni845xI2cScriptAddressWrite](#) and [ni845xI2cScriptAddressRead](#) to ACK poll until an acknowledge (ACK) is detected or the timeout is reached. If enabled, ACK polling occurs after either a start condition or a restart condition. The ACK poll timeout value persists throughout the script and is reset at the end of the script.

## ni845xI2cScriptAddressRead

---

### Purpose

Adds an I<sup>2</sup>C Script Address+Read command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I<sup>2</sup>C bus. The direction bit is internally set to 1 for read.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script Address+Read command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C 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 I<sup>2</sup>C bus using an I<sup>2</sup>C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I<sup>2</sup>C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, the NI 845x device ACK polls as set in [ni845xI2cScriptAckPollTimeout](#). (Refer to the [I<sup>2</sup>C ACK Polling](#) section in Chapter 1, *Introduction*, for more information about ACK polling.)

## ni845xI2cScriptAddressWrite

---

### Purpose

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

### Format

```
int32 ni845xI2cScriptAddressWrite (
    NiHandle ScriptHandle,
    UInt8     Address
);
```

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`UInt8 Address`

The 7-bit I<sup>2</sup>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<sup>2</sup>C Script Address+Write command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes a 7-bit address to the I<sup>2</sup>C bus connected to the I<sup>2</sup>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<sup>2</sup>C bus using an I<sup>2</sup>C script start command. It clocks out the 7-bit address and direction bit and then waits for a slave device on the I<sup>2</sup>C bus to acknowledge or not acknowledge the address. If a slave does not acknowledge the address, the NI 845x device ACK polls as set in [ni845xI2cScriptAckPollTimeout](#). (Refer to the [I<sup>2</sup>C ACK Polling](#) section in Chapter 1, *Introduction*, for more information about ACK polling.)

## ni845xI2cScriptClockRate

---

### Purpose

Adds an I<sup>2</sup>C Script Clock Rate command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C clock rate.

### Format

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

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt16 ClockRate`

The I<sup>2</sup>C clock rate in kilohertz. Refer to Chapter 3, *NI USB-845x Hardware Overview*, to determine which clock rates your NI 845x 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 I<sup>2</sup>C Script Clock Rate command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C clock rate for the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. The NI 845x 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 I<sup>2</sup>C script.

### Format

```
int32 ni845xI2cScriptClose (NiHandle ScriptHandle);
```

### Inputs

NiHandle 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<sup>2</sup>C Script Delay command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command adds a millisecond delay after the previous I<sup>2</sup>C script command.

### Format

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

### Inputs

`NiHandle 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<sup>2</sup>C Script Delay command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command adds a delay after the previous I<sup>2</sup>C script command in milliseconds.

## ni845xI2cScriptDioConfigureLine

---

### Purpose

Adds an I<sup>2</sup>C Script DIO Configure Line command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command configures a DIO line on an NI 845x device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Configure Line command to an I<sup>2</sup>C 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).



## ni845xI2cScriptDioConfigurePort

---

### Purpose

Adds an I<sup>2</sup>C Script DIO Configure Port command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command configures a DIO port on an NI 845x device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Configure Port command to an I<sup>2</sup>C 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the port to configure. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).

## ni845xI2cScriptDioReadLine

---

### Purpose

Adds an I<sup>2</sup>C Script DIO Read Line command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command reads from a DIO line on an NI 845x device.

### Format

```
int32 ni845xI2cScriptDioReadLine (
    NiHandle ScriptHandle,
    UInt8     PortNumber,
    UInt8     LineNumber,
    UInt32*   pScriptReadIndex
);
```

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Read command to an I<sup>2</sup>C script referenced by `ScriptHandle`. Use this command to read one line, specified by `LineNumber`, of a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 I<sup>2</sup>C Script DIO Read Port command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command reads from a DIO port on an NI 845x device.

### Format

```
int32 ni845xI2cScriptDioReadPort (
    NiHandle ScriptHandle,
    UInt8 PortNumber,
    UInt32 * pScriptReadIndex
);
```

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Read Port command to an I<sup>2</sup>C script referenced by `ScriptHandle`. Use this command to read all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 I<sup>2</sup>C Script DIO Write Line command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes to a DIO line on an NI 845x device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Write Line command to an I<sup>2</sup>C 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).

## ni845xI2cScriptDioWritePort

---

### Purpose

Adds an I<sup>2</sup>C Script DIO Write Port command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes to a DIO port on an NI 845x device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script DIO Write Port command to an I<sup>2</sup>C script referenced by `ScriptHandle`. Use this command to write all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).

## ni845xI2cScriptPullupEnable

---

### Purpose

Adds an I<sup>2</sup>C Script Pullup Enable command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command enables or disables the onboard pullup resistors on an NI 845x device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C Script Pullup Enable command to an I<sup>2</sup>C script referenced by `ScriptHandle`. Use this command to enable or disable the onboard pullup resistors for I<sup>2</sup>C operations. The pullup resistors pull SDA and SCL up to [ni845xSetIoVoltageLevel](#).



## ni845xI2cScriptExtractReadData

---

### Purpose

Extracts the desired read data from an I<sup>2</sup>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`.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C script, indicated by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each I<sup>2</sup>C script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index.

## ni845xI2cScriptExtractReadDataSize

---

### Purpose

Retrieves the read data size from an I<sup>2</sup>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`.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C script, indicated by `ScriptHandle`, which has been processed by `ni845xI2cScriptRun`. Each I<sup>2</sup>C script read command (`ni845xI2cScriptRead`, `ni845xI2cScriptDioReadPort`, `ni845xI2cScriptDioReadLine`) returns a script read index.

## ni845xI2cScriptHSEnable

---

### Purpose

Adds an I<sup>2</sup>C Script HS Enable command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command enables or disables High Speed mode on an NI 845x device.

### Format

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

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt8 HSEnable`

Enables or disables I<sup>2</sup>C High Speed mode. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), to determine whether your NI 845x 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 I<sup>2</sup>C Script High Speed enable command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C High Speed enable status for the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. If the hardware does not support High Speed mode, the NI-845x driver generates an error.

## ni845xI2cScriptHSMasterCode

---

### Purpose

Adds an I<sup>2</sup>C Script High Speed Master Code command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C High Speed master code.

### Format

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

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt8 HSMasterCode`

The lower 3 bits of the I<sup>2</sup>C 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 I<sup>2</sup>C Script HS Master Code command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes a master code to the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. This command assumes a start condition previously has been issued to the I<sup>2</sup>C bus using an I<sup>2</sup>C 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 I<sup>2</sup>C 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 I<sup>2</sup>C Script High Speed Clock Rate command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C High Speed clock rate.

### Format

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

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt16 HSClockRate`

Specifies the I<sup>2</sup>C High Speed clock rate. Refer to Appendix A, *NI USB-845x Hardware Specifications*, to determine which High Speed clock rates your NI 845x 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 I<sup>2</sup>C Script High Speed Clock Rate command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command sets the I<sup>2</sup>C High Speed clock rate for the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. The NI 845x 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.

## ni845xI2cScriptIssueStart

---

### Purpose

Adds an I<sup>2</sup>C Script Issue Start command to an I<sup>2</sup>C script indicated by `ScriptHandle`. This command issues a start condition on the I<sup>2</sup>C bus.

### Format

```
int32 ni845xI2cScriptIssueStart (  
    NiHandle ScriptHandle  
);
```

### Inputs

`NiHandle 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 I<sup>2</sup>C Script Issue Start command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command issues a start condition on the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. This command first waits for the I<sup>2</sup>C bus to be free. If the I<sup>2</sup>C bus is not free within the one second timeout of your NI 845x device, an error is returned when [ni845xI2cScriptRun](#) is executed. If the bus is free before the timeout, the NI 845x device issues the start condition on the I<sup>2</sup>C bus connected to the specified I<sup>2</sup>C port. This command should also be used to issue a restart condition within an I<sup>2</sup>C transaction.

## ni845xI2cScriptIssueStop

---

### Purpose

Adds an I<sup>2</sup>C Script Issue Stop command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command issues a stop condition on the I<sup>2</sup>C bus.

### Format

```
int32 ni845xI2cScriptIssueStop (  
    NiHandle ScriptHandle  
);
```

### Inputs

`NiHandle 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 I<sup>2</sup>C Script Issue Stop command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command issues a stop condition on the I<sup>2</sup>C bus connected to the I<sup>2</sup>C port you specify when you use [ni845xI2cScriptRun](#) to execute the script. Per the NXP I<sup>2</sup>C specification, you must terminate all I<sup>2</sup>C transactions with a stop condition.

## ni845xI2cScriptOpen

---

### Purpose

Opens an empty I<sup>2</sup>C script to begin adding commands to.

### Format

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

### Inputs

None.

### Outputs

```
NiHandle * 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-845x I<sup>2</sup>C Scripting API. Pass the reference to I<sup>2</sup>C script functions to create the script. Then, call [ni845xI2cScriptRun](#) to execute your script on your NI 845x device. After you finish executing your script, use [ni845xI2cScriptClose](#) to delete the script.



## ni845xI2cScriptRead

---

### Purpose

Adds an I<sup>2</sup>C Script Read command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command reads an array of data from an I<sup>2</sup>C slave device.

### Format

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

### Inputs

`NiHandle 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 I<sup>2</sup>C 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 I<sup>2</sup>C Script Read command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command reads an array of data from a device connected to the I<sup>2</sup>C 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 I<sup>2</sup>C bus using prior I<sup>2</sup>C script commands. It clocks in `NumBytesToRead` bytes from the I<sup>2</sup>C slave device, acknowledging each byte up to the last one. Depending on the type of I<sup>2</sup>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 `NotAcknowledgeLastByte` input. To obtain the data read from the specified I<sup>2</sup>C 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 I<sup>2</sup>C script referenced by `ScriptHandle` to an empty state.

### Format

```
int32 ni845xI2cScriptReset (NiHandle ScriptHandle);
```

### Inputs

NiHandle 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<sup>2</sup>C script to the desired NI 845x device, which then interprets and runs it.

### Format

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

### Inputs

NiHandle ScriptHandle

The script handle returned from [ni845xI2cScriptOpen](#).

NiHandle 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<sup>2</sup>C script indicated by `ScriptHandle` on the device indicated by `DeviceHandle`. You must first create an I<sup>2</sup>C script using the I<sup>2</sup>C scripting commands. Next, pass the script handle into `ScriptHandle`. If you have multiple NI 845x devices installed in your system, you can select which device to write your I<sup>2</sup>C script to by passing its handle into `DeviceHandle`. If your NI 845x device supports multiple I<sup>2</sup>C ports, you can also select which port to write your I<sup>2</sup>C script to. For single I<sup>2</sup>C port NI 845x devices, you must use the default port (0). In this way, you can create one script to run on various NI 845x devices, on various I<sup>2</sup>C ports within those devices. `ni845xI2cScriptRun` loads and executes your I<sup>2</sup>C script on the NI 845x device and I<sup>2</sup>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`.

## ni845xI2cScriptUsDelay

---

### Purpose

Adds an I<sup>2</sup>C Script  $\mu$ s Delay command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command adds a microsecond delay after the previous I<sup>2</sup>C script command.

### Format

```
int32 ni845xI2cScriptUsDelay (  
    uInt32 ScriptHandle,  
    uInt16 Delay  
);
```

### Inputs

`uInt32 ScriptHandle`

The script handle returned from [ni845xI2cScriptOpen](#).

`uInt16 Delay`

The desired delay in microseconds.

### 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 `ni845xI2cScriptUsDelay` to add an I<sup>2</sup>C Script  $\mu$ s Delay command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command adds a delay after the previous I<sup>2</sup>C script command in microseconds.

## ni845xI2cScriptWrite

---

### Purpose

Adds an I<sup>2</sup>C Script Write command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes an array of data to an I<sup>2</sup>C slave device.

### Format

```
int32 ni845xI2cScriptWrite (
    NiHandle ScriptHandle,
    uint32 WriteSize,
    uint8 * pWriteData
);
```

### Inputs

`NiHandle 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 I<sup>2</sup>C Script Write command to an I<sup>2</sup>C script referenced by `ScriptHandle`. This command writes an array of data to an I<sup>2</sup>C slave device connected to the I<sup>2</sup>C 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 I<sup>2</sup>C bus using prior I<sup>2</sup>C script commands. It clocks the `pWriteData` array into the I<sup>2</sup>C 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.

---

# Using the NI-845x I<sup>2</sup>C Slave API

This chapter helps you get started with the I<sup>2</sup>C Slave API.

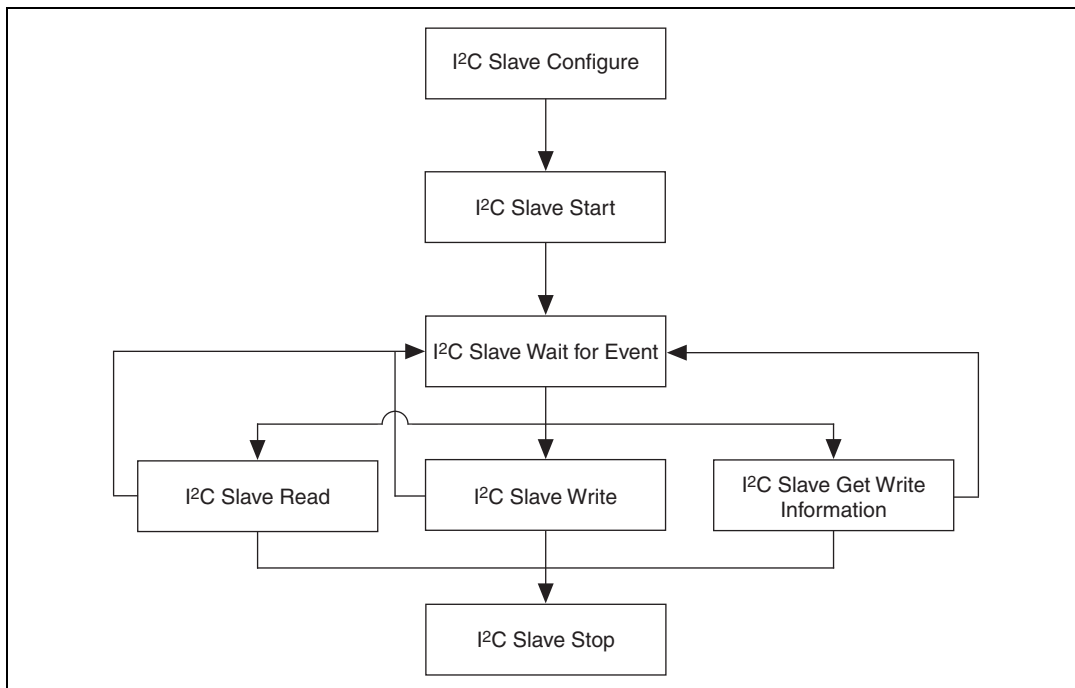
## I<sup>2</sup>C Slave Programming Model

---

The I<sup>2</sup>C Slave API allows for an event-based programming model to emulate many of off-the-shelf I<sup>2</sup>C devices.

When you use the I<sup>2</sup>C Slave API, the first step is to create an I<sup>2</sup>C Slave configuration to describe the communication requirements between the NI 845x device and the I<sup>2</sup>C master device. To make an I<sup>2</sup>C Slave configuration, create an I<sup>2</sup>C Slave configuration reference and set the appropriate properties as desired. You can then wait for the I<sup>2</sup>C master device to begin communication.

The diagram in Figure 8-1 describes the programming model for the NI-845x I<sup>2</sup>C Slave API. The diagram is followed by a description of each step in the model.



**Figure 8-1.** Programming Model for I<sup>2</sup>C Slave Communication

## I<sup>2</sup>C Slave Configure

Use the [NI-845x I2C Slave Configuration Property Node](#) in LabVIEW and `ni845xI2cSlaveConfiguration*` calls in other languages to set the specific I<sup>2</sup>C configuration that describes the characteristics of the slave to emulate.

## I<sup>2</sup>C Slave Start

Use [NI-845x I2C Slave Start.vi](#) in LabVIEW and `ni845xI2cSlaveStart` in other languages to commit the slave configuration and begin slave operation.



## I<sup>2</sup>C Slave Wait For Event

Use **NI-845x I2C Slave Wait for Event.vi** in LabVIEW and `ni845xI2cSlaveWaitForEvent` in other languages to wait for an event from the I<sup>2</sup>C Slave and retrieve pertinent event information.

## I<sup>2</sup>C Slave Read

Use **NI-845x I2C Slave Read.vi** in LabVIEW and `ni845xI2cSlaveRead` in other languages to process a data available event or a command event and read the data read from the I<sup>2</sup>C master.

## I<sup>2</sup>C Slave Write

Use **NI-845x I2C Slave Write.vi** in LabVIEW and `ni845xI2cSlaveWrite` in other languages to process a data requested event and provide the data to write to satisfy the I<sup>2</sup>C master's read requests.

## I<sup>2</sup>C Slave Get Write Information

Use **NI-845x I2C Slave Get Write Info.vi** in LabVIEW and `ni845xI2cSlaveGetWriteInfo` in other languages to process a write complete event to retrieve information about the completed write to the I<sup>2</sup>C master.

## I<sup>2</sup>C Slave Stop

Use **NI-845x I2C Slave Stop.vi** in LabVIEW and `ni845xI2cSlaveStop` in other languages to stop slave operations.

---

# NI-845x I<sup>2</sup>C Slave API for LabVIEW

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



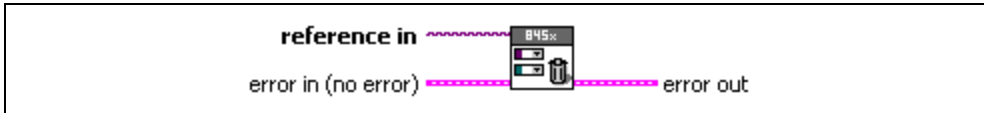
**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

# General Device

## NI-845x Close Reference.vi

### Purpose

Closes a previously opened reference.



### Inputs



**reference in** is a reference to an NI 845x device, I<sup>2</sup>C configuration, I<sup>2</sup>C Slave configuration, SPI configuration, SPI stream configuration, I<sup>2</sup>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.

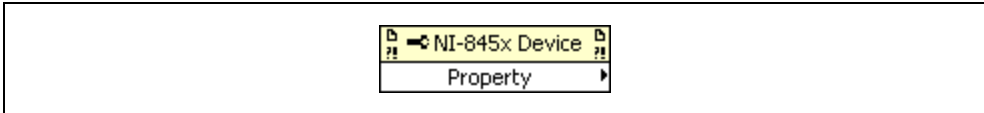
## Description

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

## NI-845x Device Property Node

### Purpose

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



### Inputs



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



### 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<sup>2</sup>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<sup>2</sup>C Pullup Enable

The **I<sup>2</sup>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.



### Timeout (ms)

The **Timeout (ms)** property sets the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.



**Note** It is highly recommended to set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

The default of this property is 30000 (30 seconds).

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

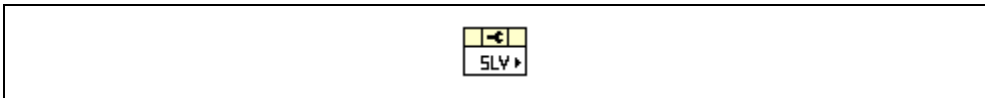


# Configuration

## NI-845x I<sup>2</sup>C Slave Configuration Property Node

### Purpose

A property node with the NI-845x I<sup>2</sup>C Slave Configuration class preselected. This property node allows you to query and modify I<sup>2</sup>C Slave configuration properties of your NI 845x device.



### Inputs



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**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<sup>2</sup>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 I<sup>2</sup>C Slave Configuration Property Node**.



### Address

Specifies the address of the I<sup>2</sup>C slave. The NXP I<sup>2</sup>C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C Specification and defines an address for a 7-bit device as a 7-bit value. The I<sup>2</sup>C Slave API does not support 10-bit addresses.

The default value of this property is 0. The default value is not recommended for use as it is reserved by the NXP I<sup>2</sup>C Standard as the general call address.



### Command Buffer Size

Specifies the number of bytes to be used as a command buffer. This buffer is used to store the first bytes when the master initiates a write operation. You can use this buffer if the slave device being emulated requires a command to be prepended before data can be written to it. When this buffer is filled or if the write operation is completed, a [CmdEvent](#) is generated.

Providing a command buffer size of 0 disables the command buffer.

The default value of this property is 0.



### Data Buffer Size

Specifies the number of bytes to be used as a data buffer. This buffer is used to store data when the master performs a write operation to the slave. When the command buffer is enabled, only the data after the command is stored. If this buffer is full or if the write operation is completed, a [DataAvailEvent](#) is generated.

The default value of this property is 128.



### Auto Removal Timeout

Specifies the amount of time in milliseconds to remove the slave from the I<sup>2</sup>C bus after the slave has completed reading data from the master.

- 0 The slave does not remove itself from the bus after reading data from the master.
- > 0 The slave removes itself from the bus for the amount of time specified by the property.
- < 0 The slave removes itself from the bus and immediately stops operation. It will not return to the I<sup>2</sup>C bus until **NI-845x I2C Slave Start.vi** is called.

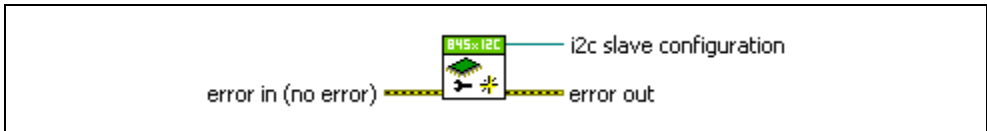
You can use this timeout if the slave device being emulated requires time to process received data. An example of this is an EEPROM that needs to simulate the write cycle timing for a page of data.

The default value of this property is 0.

## NI-845x I<sup>2</sup>C Slave Create Configuration Reference.vi

### Purpose

Creates a new NI-845x I<sup>2</sup>C Slave 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 slave configuration** is a reference to the newly created NI-845x I<sup>2</sup>C Slave 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.

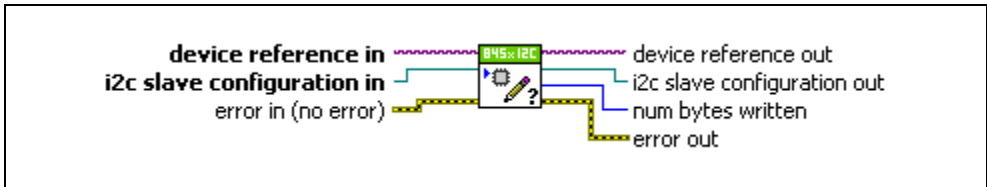
## Description

Use **NI-845x I<sup>2</sup>C Slave Create Configuration Reference.vi** to create a new configuration to use with the NI-845x I<sup>2</sup>C Slave API. Pass the reference to a property node to make the configuration match the desired settings. Then pass the configuration to the I<sup>2</sup>C Slave functions to execute them. After you finish communication, pass the reference into a new property node to reconfigure it or use **NI-845x Close Reference.vi** to delete the configuration.

## NI-845x I<sup>2</sup>C Slave Get Write Info.vi

### Purpose

Processes a `WriteCompleteEvent`.



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**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 845x device after this VI runs.



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave configuration after this VI runs.



**num bytes written** indicates the number of bytes transferred from the slave to the master during the previous event.



**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 I<sup>2</sup>C Slave Get Write Info.vi** to process a [WriteCompleteEvent](#). The number of bytes written specified indicates the number of bytes written to the master during the previous [DataReqEvent](#).

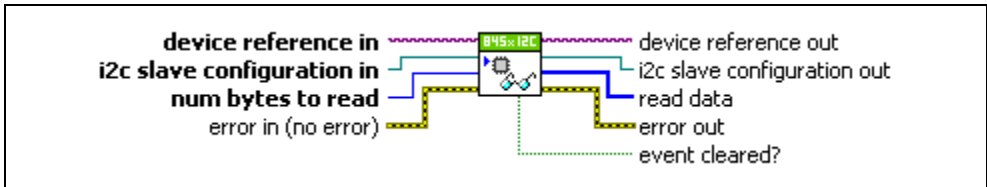


**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## NI-845x I<sup>2</sup>C Slave Read.vi

### Purpose

Processes a [DataAvailEvent](#) or a [CmdEvent](#).



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**num bytes to read** is the maximum number of bytes to read. To complete processing an event, all of the bytes from the [DataAvailEvent](#) or the [CmdEvent](#) must be read. The maximum number of bytes in an event is defined by the maximum of the properties **Data Buffer Size** and **Command Buffer Size**.



**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 845x device after this VI runs.



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave configuration after this VI runs.



**read data** is the data contained in the event. The amount of data read from the event is the minimum of **num bytes to read** and the number of bytes left in the event.



**event cleared?** indicates all data has been read from the event and no further action is required for the current event.



**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 Slave Read.vi** to process a [DataAvailEvent](#) or [CmdEvent](#). The data received from the master is contained in read data.

For the [CmdEvent](#), the maximum amount of data is defined by **Command Buffer Size**. For the [DataAvailEvent](#), the maximum amount of data is defined by **Data Buffer Size**.

The event is completely processed when all data has been read from the event.

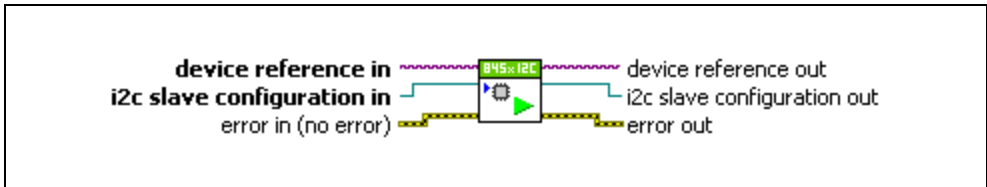


**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## NI-845x I<sup>2</sup>C Slave Start.vi

### Purpose

Commits the I<sup>2</sup>C Slave configuration and places the I<sup>2</sup>C Slave on the bus.



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**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 845x device after this VI runs.



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave 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 Slave Start.vi** to commit the I<sup>2</sup>C Slave configuration and place the emulated slave on the I<sup>2</sup>C bus. Once the emulated slave is on the I<sup>2</sup>C bus, the address specified in the configuration is acknowledged when addressed by the master. The emulated slave begins generating events that can be read using **NI-845x I2C Slave Wait for Event.vi**.

Before using **NI-845x I2C Slave Start.vi**, ensure that the configuration parameters specified in **i2c slave configuration in** are correct for the slave you want to emulate.

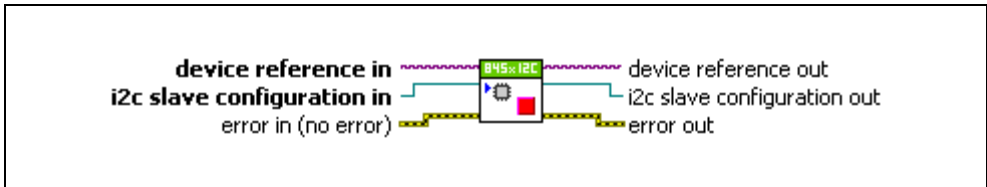


**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## NI-845x I<sup>2</sup>C Slave Stop.vi

### Purpose

Removes the I<sup>2</sup>C Slave from the bus and clears any pending events.



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**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 845x device after this VI runs.



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave 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 I<sup>2</sup>C Slave Stop.vi** to remove the emulated slave from the I<sup>2</sup>C bus and clear any pending events.

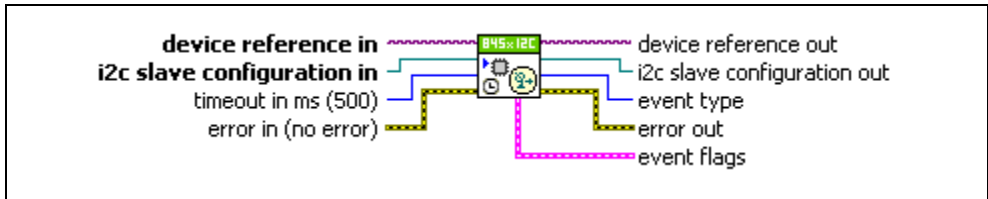


**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## NI-845x I<sup>2</sup>C Slave Wait for Event.vi

### Purpose

Waits for an event to be generated from the I<sup>2</sup>C Slave.



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**timeout in ms** is the amount of time in milliseconds to wait for an event. When a timeout occurs, `NoEvent` is returned.



**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



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



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave configuration after this VI runs.



**event type** is the event generated from the I<sup>2</sup>C Slave. Reading the event does not complete the event. Each event generated requires further action to process. **event type** uses the following values:

#### NoEvent

No event was generated and a timeout occurred.

No actions are required to process this event.

#### CmdEvent

A command is available to be read from the slave. This is due to the master initiating a write operation and filling the slave's command buffer with the number of bytes configured in **Command Buffer Size** or completing the write operation with a RESTART or STOP condition.

To process this event, **NI-845x I2C Slave Read.vi** must be called to read the data written by the I<sup>2</sup>C master.

#### DataAvailEvent

Data is available to be read from the slave. This is due to the master initiating a write operation and filling the slave's data buffer with the number of bytes configured in **Data Buffer Size**, or the write operation was completed with a RESTART or STOP condition on the I<sup>2</sup>C bus.

To process this event, **NI-845x I2C Slave Read.vi** must be called to read the data written by the I<sup>2</sup>C master.

#### DataReqEvent

Data is requested to be written to the master. This is due to the master requesting data to be read from the emulated slave.

To process this event, **NI-845x I2C Slave Write.vi** must be called to provide data to the slave to write to the I<sup>2</sup>C master.

#### WriteCompleteEvent

The slave device has completed writing data to the master. This is due to the master reading a byte from the slave and sending a NACK, or all data from the previous `DataReqEvent` has been transferred and the master is requesting more data. When this event is generated, any additional write data present on the device is cleared.

To process this event, **NI-845x I2C Slave Get Write Info.vi** must be called.



**event flags** describes the conditions surrounding the creation of the event.



**IO Start** indicates this is the first event since the I/O operation began.



**IO Complete** indicates that the I/O operation has completed for any reason including a STOP or RESTART condition on the I<sup>2</sup>C bus.



**IO Complete Stop** indicates that the I/O operation has completed due to a STOP condition on the I<sup>2</sup>C bus.



**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 I<sup>2</sup>C Slave Wait For Event.vi** to read the latest event that has occurred on the emulated slave. While in operation, only one event is active at a time and must be fully processed by the appropriate handler. Each event is mapped to a single handler as described in the following table.

Event	Handler
NoEvent	N/A
CmdEvent	<b>NI-845x I<sup>2</sup>C Slave Read.vi</b>
DataAvailEvent	<b>NI-845x I<sup>2</sup>C Slave Read.vi</b>
DataReqEvent	<b>NI-845x I<sup>2</sup>C Slave Write.vi</b>
WriteCompleteEvent	<b>NI-845x I<sup>2</sup>C Slave Get Write Info.vi</b>

**NI-845x Slave Wait For Event.vi** returns an error if a previous event has not been fully processed.



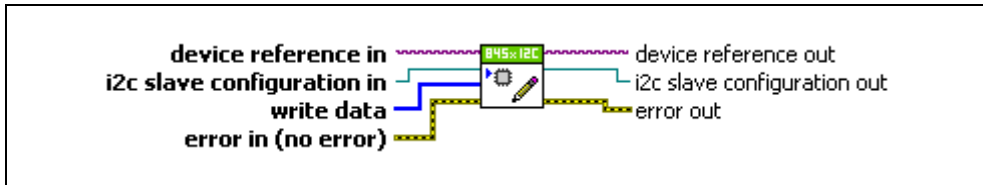
**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.



## NI-845x I<sup>2</sup>C Slave Write.vi

### Purpose

Processes a [DataReqEvent](#).



### Inputs



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



**i2c slave configuration in** is a reference to a specific I<sup>2</sup>C Slave configuration that describes the characteristics of the slave to emulate.



**write data** is the buffer of data to be written to the slave. Each time the master requests a data byte, one byte from the write data is transferred.



**Note** The maximum amount of write data that is available to the slave is 65,535 bytes per event. Any additional data is ignored.



**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 845x device after this VI runs.



**i2c slave configuration out** is a reference to the I<sup>2</sup>C Slave 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 Slave Write.vi** to process a [DataReqEvent](#). Data provided is used to satisfy read requests from the master. When all provided write data has been transmitted, a [WriteCompleteEvent](#) is generated followed by a [DataReqEvent](#) if the master is requesting more data.

On completion of the master's read sequence, any extra write data pending transmission from the slave is discarded.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

---

# NI-845x I<sup>2</sup>C Slave API for C

This chapter lists the functions for the NI-845x I<sup>2</sup>C Slave API and describes the format, purpose, and parameters for each function. The functions are listed alphabetically in three categories: general device, configuration, and slave.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## Section Headings

---

The NI-845x I<sup>2</sup>C Slave 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-845x I<sup>2</sup>C Slave 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

Data Type	Purpose
int8	8-bit signed integer
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')
NiHandle	Operating system independent handle

## List of Functions

The following table contains an alphabetical list of the NI-845x I<sup>2</sup>C Slave API for C functions.

Function	Purpose
<a href="#">ni845xClose</a>	Closes a previously opened NI 845x device.
<a href="#">ni845xCloseFindDeviceHandle</a>	Closes the handles created by <a href="#">ni845xFindDevice</a> .
<a href="#">ni845xDeviceLock</a>	Locks NI 845x devices for access by a single thread.
<a href="#">ni845xDeviceUnlock</a>	Unlocks NI 845x devices.
<a href="#">ni845xFindDevice</a>	Finds an NI 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using <a href="#">ni845xFindDeviceNext</a> .

Function	Purpose
<code>ni845xFindDeviceNext</code>	Finds subsequent devices after <code>ni845xFindDevice</code> has been called.
<code>ni845xI2cSetPullupEnable</code>	Modifies the voltage output from a DIO port on an NI 845x device.
<code>ni845xI2cSlaveConfigurationClose</code>	Closes an NI-845x I <sup>2</sup> C Slave I/O configuration.
<code>ni845xI2cSlaveConfigurationGetAddress</code>	Retrieves the configuration's address.
<code>ni845xI2cSlaveConfigurationGetAutoRemovalTimeout</code>	Retrieves the configuration's auto removal timeout.
<code>ni845xI2cSlaveConfigurationGetCommandBufferSize</code>	Retrieves the configuration's command buffer size.
<code>ni845xI2cSlaveConfigurationGetDataBufferSize</code>	Retrieves the configuration's data buffer size.
<code>ni845xI2cSlaveConfigurationOpen</code>	Creates a new NI-845x I <sup>2</sup> C Slave configuration.
<code>ni845xI2cSlaveConfigurationSetAddress</code>	Sets the configuration's address.
<code>ni845xI2cSlaveConfigurationSetAutoRemovalTimeout</code>	Sets the configuration's auto removal timeout.
<code>ni845xI2cSlaveConfigurationSetCommandBufferSize</code>	Sets the configuration's command buffer size.
<code>ni845xI2cSlaveConfigurationSetDataBufferSize</code>	Sets the configuration's data buffer size.
<code>ni845xI2cSlaveGetWriteInfo</code>	Processes a <code>Write Complete Event</code> .
<code>ni845xI2cSlaveRead</code>	Processes a <code>Data Available Event</code> or a <code>Command Event</code> .

Function	Purpose
<code>ni845xI2cSlaveStart</code>	Commits the I <sup>2</sup> C Slave configuration and places the I <sup>2</sup> C Slave on the bus.
<code>ni845xI2cSlaveStop</code>	Removes the I <sup>2</sup> C Slave from the bus and clears any pending events.
<code>ni845xI2cSlaveWaitForEvent</code>	Waits for an event to be generated from the I <sup>2</sup> C Slave.
<code>ni845xI2cSlaveWrite</code>	Processes a <a href="#">Data Requested Event</a> .
<code>ni845xOpen</code>	Opens an NI 845x device for use with various write, read, and device property functions.
<code>ni845xSetIoVoltageLevel</code>	Sets the voltage level of the NI-845x I/O pins (DIO/SPI/VioRef).
<code>ni845xSetTimeout</code>	Sets the global timeout value.
<code>ni845xStatusToString</code>	Converts a status code into a descriptive string.

# General Device

---

## ni845xClose

---

### Purpose

Closes a previously opened NI 845x device.

### Format

```
int32 ni845xClose(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 (  
    NiHandle FindDeviceHandle  
);
```

### Inputs

```
NiHandle 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 845x devices for access by a single thread.

### Format

```
int32 ni845xDeviceLock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x devices.

### Format

```
int32 ni845xDeviceUnlock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using [ni845xFindDeviceNext](#).

### Format

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

### Inputs

None.

### Outputs

```
char * pFirstDevice
```

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

```
NiHandle * 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 845x 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 845x device and the number of NI 845x 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 845x 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 (
    NiHandle FindDeviceHandle,
    char *   pNextDevice
);
```

### Inputs

NiHandle FindDeviceHandle

Describes a find list. [ni845xFindDevice](#) creates this parameter.

### Outputs

char \* pNextDevice

A pointer to the string containing the next NI 845x 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 845x device for use with various write, read, and device property functions.

### Format

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

### Inputs

```
char * pResourceName
```

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

### Outputs

```
NiHandle * 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 845x 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 845x device.

### Format

```
int32 ni845xSetIoVoltageLevel (
    NiHandle DeviceHandle,
    UInt8 VoltageLevel
);
```

### Inputs

NiHandle 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 845x device. The board reference voltage is used for SPI, I<sup>2</sup>C, 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 845x device.

### Format

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

### Inputs

NiHandle 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 I<sup>2</sup>C operations. The pullup resistors pull SDA and SCL up to [ni845xSetIoVoltageLevel](#).



## ni845xSetTimeout

---

### Purpose

Modifies the global timeout for operations when using an NI 845x device.

### Format

```
int32 ni845xSetTimeout (
    NiHandle DeviceHandle,
    uInt32 Timeout
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

uInt32 Timeout

The timeout value in milliseconds. The minimum timeout is 1000 ms (1 second).

### 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 `ni845xI2cSetTimeout` to set the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.

The default of this property is 30000 (30 seconds).



**Note** You should set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

## 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-845x 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-845x 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-845x function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845x use of this status.

## NI-845x 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-845x function. If an error is detected, you should close all NI-845x 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<sup>2</sup>C 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

---

## ni845xI2cSlaveConfigurationClose

---

### Purpose

Closes an I<sup>2</sup>C Slave I/O configuration.

### Format

```
int32 ni845xI2cSlaveConfigurationClose (  
    NiHandle ConfigurationHandle  
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### 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 `ni845xI2cSlaveConfigurationClose` to close a configuration.

## ni845xI2cSlaveConfigurationGetAddress

---

### Purpose

Retrieves the configuration address.

### Format

```
int32 ni845xI2cSlaveConfigurationGetAddress (  
    NiHandle ConfigurationHandle,  
    uInt16 * pAddress  
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### Outputs

uInt16 \* pAddress

A pointer to an unsigned 16-bit integer to store the configuration's 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 [ni845xI2cSlaveConfigurationGetAddress](#) to retrieve the configuration's slave address.

## ni845xI2cSlaveConfigurationGetAutoRemovalTimeout

---

### Purpose

Retrieves the configuration's auto removal timeout.

### Format

```
int32 ni845xI2cSlaveConfigurationGetAutoRemovalTimeout (
    NiHandle ConfigurationHandle,
    int32 * pAutoRemovalTimeout
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### Outputs

int32 \* pAutoRemovalTimeout

A pointer to a signed 32-bit integer to store the auto removal timeout.

### 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 `ni845xI2cSlaveConfigurationGetAutoRemovalTimeout` to retrieve the configuration's autoremoval timeout.

## ni845xI2cSlaveConfigurationGetCommandBufferSize

---

### Purpose

Retrieves the configuration's command buffer size.

### Format

```
int32 ni845xI2cSlaveConfigurationGetCommandBufferSize (
    NiHandle ConfigurationHandle,
    uInt16 * pCommandBufferSize
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### Outputs

uInt16 \* pCommandBufferSize

A pointer to an unsigned 16-bit integer to store the command buffer 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 [ni845xI2cSlaveConfigurationGetCommandBufferSize](#) to retrieve the configuration's command buffer size.

## ni845xI2cSlaveConfigurationGetDataBufferSize

---

### Purpose

Retrieves the configuration's data buffer size.

### Format

```
int32 ni845xI2cSlaveConfigurationGetDataBufferSize (  
    NiHandle ConfigurationHandle,  
    uInt16 * pDataBufferSize  
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### Outputs

uInt16 \* pDataBufferSize

A pointer to an unsigned 16-bit integer to store the data buffer 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 `ni845xI2cSlaveConfigurationGetDataBufferSize` to retrieve the configuration's data buffer size.



## ni845xI2cSlaveConfigurationOpen

---

### Purpose

Creates a new NI-845x I<sup>2</sup>C Slave configuration.

### Format

```
int32 ni845xI2cSlaveConfigurationOpen (  
    NiHandle * pConfigurationHandle  
);
```

### Inputs

NiHandle \* pConfigurationHandle

The a pointer to an NiHandle to store the created configuration handle. This must not be NULL.

### 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 this function to create a new configuration to use with the NI-845x I<sup>2</sup>C Slave API. Pass the handles to the `ni845xI2cSlaveConfigurationSet*` series of functions to modify the configuration properties. Then, pass the configuration to the I<sup>2</sup>C slave functions to execute them. After you finish emulating the slave, pass the handle to the `ni845xI2cSlaveConfigurationSet*` series of functions to reconfigure it or use [ni845xI2cSlaveConfigurationClose](#) to delete the configuration.

## ni845xI2cSlaveConfigurationSetAddress

---

### Purpose

Sets the configuration's I<sup>2</sup>C slave address.

### Format

```
int32 ni845xI2cSlaveConfigurationSetAddress (
    NiHandle ConfigurationHandle,
    uInt16 Address
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt16 Address

Specifies the address of the I<sup>2</sup>C Slave. The NXP I<sup>2</sup>C Specification defines a 7-bit slave address and a direction bit. During the address phase of an I<sup>2</sup>C transaction, these values are sent across the bus as one byte (slave address in bits 7–1, direction in bit 0). The NI-845x software follows the convention used in the NXP I<sup>2</sup>C Specification and defines an address for a 7-bit device as a 7-bit value. The I<sup>2</sup>C Slave does not support 10-bit addresses.

The default value of this property is 0. The default value is not recommended for use, as it is reserved by the NXP I<sup>2</sup>C Standard as the general call address.

### 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 `ni845xI2cSlaveConfigurationSetAddress` to set the I<sup>2</sup>C slave address. This is a 7-bit number; do not include a direction bit.

## ni845xI2cSlaveConfigurationSetAutoRemovalTimeout

---

### Purpose

Sets the configuration's auto removal timeout.

### Format

```
int32 ni845xI2cSlaveConfigurationSetAutoRemovalTimeout (
    NiHandle ConfigurationHandle,
    int32     AutoRemovalTimeout
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

int32 AutoRemovalTimeout

A signed 32-bit integer to store the auto removal timeout.

### 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 `ni845xI2cSlaveConfigurationSetAutoRemovalTimeout` to set the configuration's autoremoval timeout in milliseconds. The timeout removes the slave from the I<sup>2</sup>C bus after the slave has completed reading from the master.

Valid values for `AutoRemovalTimeout` include:

- 0            The slave does not remove itself from the bus after reading data from the master.
- > 0         The slave removes itself from the bus for the amount of time specified by the property.
- < 0         The slave removes itself from the bus and immediately stops operation. It will not return to the I<sup>2</sup>C bus until [ni845xI2cSlaveStart](#) is called.

You can use this timeout if the slave device being emulated requires time to processes received data. An example of this is an EEPROM that needs to simulate the write cycle timing for a page of data.

The default value of this property is 0.

## ni845xI2cSlaveConfigurationSetCommandBufferSize

---

### Purpose

Sets the configuration's command buffer size.

### Format

```
int32 ni845xI2cSlaveConfigurationSetCommandBufferSize (
    NiHandle ConfigurationHandle,
    uInt16 CommandBufferSize
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt16 CommandBufferSize

Specifies the number of bytes to be used as a command buffer. This buffer is used to store the first bytes when the master initiates a write operation. You can use this buffer if the slave device being emulated requires a command to be prepended before data can be written to it. When this buffer is filled, a [Command Event](#) is generated.

Providing a command buffer size of 0 disables the command buffer.

The default value of this property 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 `ni845xI2cSlaveConfigurationSetCommandBufferSize` to set the command buffer size.

## ni845xI2cSlaveConfigurationSetDataBufferSize

---

### Purpose

Sets the configuration's data buffer size.

### Format

```
int32 ni845xI2cSlaveConfigurationSetDataBufferSize (
    NiHandle ConfigurationHandle,
    uInt16   DataBufferSize
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt16 DataBufferSize

Specifies the number of bytes to be used as a data buffer. This buffer is used to store data when the master performs a write operation to the slave. When the command buffer is enabled, only the data after the command is stored in the data buffer. If this buffer is full or if the write operation is completed, a [Data Available Event](#) is generated.

The default value of this property is 128.

### 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 [ni845xI2cSlaveConfigurationSetDataBufferSize](#) to set the data buffer size.

# Slave

---

## ni845xI2cSlaveGetWriteInfo

---

### Purpose

Processes a [Write Complete Event](#).

### Format

```
int32 ni845xI2cSlaveGetWriteInfo (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uInt16 * pNumBytesWritten
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### Outputs

uInt16 \* pNumBytesWritten

A pointer to the number of bytes transferred from the slave to the master during the previous write operation.

### 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 [ni845xI2cSlaveGetWriteInfo](#) to process a [Write Complete Event](#). The number of bytes written specified indicates the number of bytes written to the master following the previous [Data Requested Event](#). Each [Data Requested Event](#) is paired with a [Write Complete Event](#) to provide information on the completed transfer.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## ni845xI2cSlaveRead

---

### Purpose

Processes a [Data Available Event](#) or a [Command Event](#).

### Format

```
int32 ni845xI2cSlaveRead(
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uInt16 NumBytesToRead,
    uInt16 * pReadSize,
    uInt8 * pReadData,
    bool * pEventCleared
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt16 NumBytesToRead

The maximum number of bytes to read. To complete processing an event, all bytes from the [Data Available Event](#) or the [Command Event](#) must be read. The maximum number of bytes in an event is defined by the maximum of the properties set using [ni845xI2cSlaveConfigurationSetCommandBufferSize](#) and [ni845xI2cSlaveConfigurationSetDataBufferSize](#).

### Outputs

uInt16 \* pReadSize

A pointer to the number of bytes stored in pReadData.

uInt8 \* pReadData

A pointer to a buffer that holds the data contained in the event. The amount of data read from the event is the minimum of NumBytesToRead and the number of bytes left in the event.

bool \* pEventCleared

A pointer to a Boolean value that indicates if the [Data Available Event](#) or [Command Event](#) has been processed.

## 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 `ni845xI2cSlaveRead` to process a [Data Available Event](#) or [Command Event](#). The data received from the master is contained in `pReadData`.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.



## ni845xI2cSlaveStart

---

### Purpose

Commits the I<sup>2</sup>C Slave configuration and places the I<sup>2</sup>C Slave on the bus.

### Format

```
int32 ni845xI2cSlaveStart (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### 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 `ni845xI2cSlaveStart` to commit the I<sup>2</sup>C Slave configuration and place the slave on the I<sup>2</sup>C bus. Once the slave is on the I<sup>2</sup>C bus, the address specified in the configuration is acknowledged when addressed by the master. The slave begins generating events that can be read using [ni845xI2cSlaveWaitForEvent](#). Before using `ni845xI2cSlaveStart`, ensure that the configuration parameters specified in `ConfigurationHandle` are correct for the slave you want to emulate.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## ni845xI2cSlaveStop

---

### Purpose

Removes the I<sup>2</sup>C Slave from the bus and clears any pending events.

### Format

```
int32 ni845xI2cSlaveStop (  
    NiHandle DeviceHandle,  
    NiHandle ConfigurationHandle  
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

### 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 [ni845xI2cSlaveStop](#) to remove the slave from the I<sup>2</sup>C bus and clear any pending events.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## ni845xI2cSlaveWaitForEvent

---

### Purpose

Waits for an event to be generated from the I<sup>2</sup>C slave.

### Format

```
int32 ni845xI2cSlaveWaitForEvent (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uInt32 Timeout,
    uInt8 * pEventType,
    uInt16 * pEventFlags
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt32 Timeout

The amount of time in milliseconds to wait for an event. If a timeout occurs, `pEventType` contains the value 0 indicating that no event is available.

### Outputs



**Note** The following table lists the event names and their corresponding values.

Event	Value
Command Event	<code>kNi845xI2cSlaveEventCmdEvent (0x01)</code>
Data Available Event	<code>kNi845xI2cSlaveEventDataAvailEvent (0x02)</code>
Data Requested Event	<code>kNi845xI2cSlaveEventDataReqEvent (0x03)</code>
Write Complete Event	<code>kNi845xI2cSlaveEventWriteCompleteEvent (0x04)</code>

uInt8 \* pEventType

The event generated from the I<sup>2</sup>C Slave. Reading the event does not complete the processing of the event. All events generated except `kNi845xI2cSlaveEventNoEvent` require further action to process. `pEventType` uses the following values:

- `kNi845xI2cSlaveEventNoEvent (0x00)`  
No event was generated and a timeout occurred.  
No actions are required to process this event.
- `kNi845xI2cSlaveEventCmdEvent (0x01)`  
A command is available to be read from the slave. This is due to the master initiating a write operation and filling the slave's command buffer with the number of bytes configured using `ni845xI2cSlaveConfigurationSetCommandBufferSize` or completing the write operation with a RESTART or STOP condition.  
To process this event, `ni845xI2cSlaveRead` must be called to read the data written by the I<sup>2</sup>C master.
- `kNi845xI2cSlaveEventDataAvailEvent (0x02)`  
Data is available to be read from the slave. This is due to the master initiating a write operation and filling the slave's data buffer with the number of bytes configured in `ni845xI2cSlaveConfigurationSetDataBufferSize` or the write operation was completed with a RESTART or STOP condition on the I<sup>2</sup>C bus.  
To process this event, `ni845xI2cSlaveRead` must be called to read the data written by the I<sup>2</sup>C master.
- `kNi845xI2cSlaveEventDataReqEvent (0x03)`  
Data is requested to be written to the master. This is due to the master requesting data to be read from the slave.  
To process this event, `ni845xI2cSlaveWrite` must be called to provide data to the slave to write to the I<sup>2</sup>C master.
- `kNi845xI2cSlaveEventWriteCompleteEvent (0x04)`  
The slave device has completed writing data to the master. This is due to the master reading a byte from the slave and sending a NACK or all data from the previous `Data Requested Event` has been transferred. When this event is generated, any additional write data present on the device is cleared.  
To process this event, `ni845xI2cSlaveGetWriteInfo` must be called.

uInt16 \* pEventFlags

A bit field that describes the conditions surrounding the creation of the event.

Possible flags include:

- `kNi845xI2cEventFlagIoStart (0x01)`  
Indicates that this is the first event since the I/O operation began.
- `kNi845xI2cEventFlagIoComplete (0x02)`

Indicates that the I/O operation has completed for any reason including a STOP or RESTART condition on the I<sup>2</sup>C bus.

- `kNi845xI2cEventFlagIoCompleteStop` (0x04)

Indicates that the I/O operation has completed due to a STOP condition on the I<sup>2</sup>C bus.



**Note** Multiple flags may be set at one time depending on the bus conditions that caused the event.

## 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 `ni845xI2cSlaveWaitForEvent` to read the latest event that has occurred on the slave. While in operation, only one event is active at a time and must be fully processed by the appropriate handler. Each event is mapped to a single handler as described in the table below:

Event	Handler
No Event	N/A
Command Event	<code>ni845xI2cSlaveRead</code>
Data Available Event	<code>ni845xI2cSlaveRead</code>
Data Requested Event	<code>ni845xI2cSlaveWrite</code>
Write Complete Event	<code>ni845xI2cSlaveGetWriteInfo</code>

`ni845xI2cSlaveWaitForEvent` returns an error if a previous event has not been fully processed.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

## ni845xI2cSlaveWrite

---

### Purpose

Processes a [Data Requested Event](#).

### Format

```
int32 ni845xI2cSlaveWrite (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uInt16 WriteSize,
    uInt8 * pWriteData
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xI2cSlaveConfigurationOpen](#).

uInt16 WriteSize

Size of WriteData.

uInt8 \* pWriteData

The buffer of data to be written to the slave. Each time the master requests a data byte, one byte from pWriteData is transferred.

### 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 [ni845xI2cSlaveWrite](#) to process a [Data Requested Event](#). Data provided is used to satisfy read requests from the master. When all provided write data has been transmitted, a [Write Complete Event](#) is generated. If the master requests more data, a new [Data Requested Event](#) is generated.

Upon completion of the master's read sequence, a [Write Complete Event](#) is generated, and any extra write data pending transmission from the slave is discarded.



**Note** The I<sup>2</sup>C Slave API is not available for the NI USB-8451.

---

# Using the NI-845x SPI API

This chapter helps you get started with the SPI API.

---

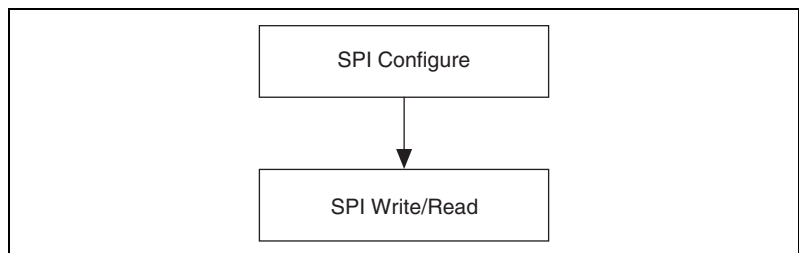
## NI-845x SPI Basic Programming Model

---

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 Scripting 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 845x 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 11-1 describes the programming model for the NI-845x 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 11-1.** NI-845x SPI API Basic Programming Model



## SPI Configure

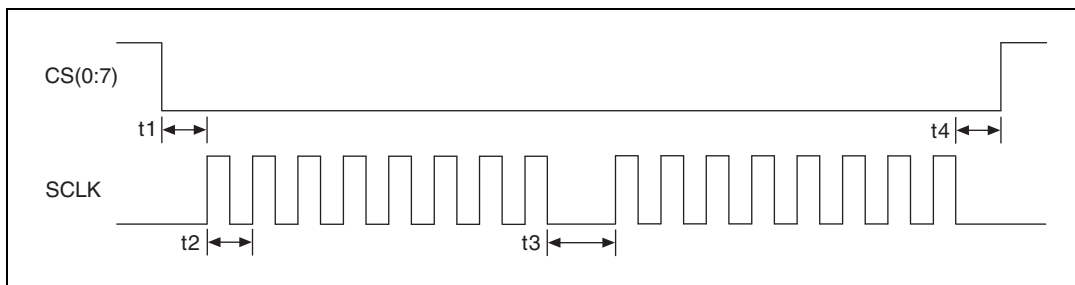
Use the **NI-845x SPI Configuration Property Node** in LabVIEW and `ni845xSpiConfiguration*` calls in other languages to set the specific SPI configuration that describes the characteristics of the device to communicate with.

## 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 11-2 and Tables 11-1 and 11-2 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 Scripting API to adjust the bus characteristics to match those of your device.



**Figure 11-2.** SPI Waveform

**Table 11-1.** 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	$\mu\text{s}$
t2	SCLK period	0.08333	20.83	$\mu\text{s}$
t3	SCLK setup time	8.5	19	$\mu\text{s}$
t4	Last SCLK edge to CS(0:7) deassertion	7.4	8.24	$\mu\text{s}$

**Table 11-2.** NI USB-8452 Basic API SPI Timing Characteristics

Symbol	Parameter	Min	Max	Units
t1	CS(0:7) assertion to first SCLK edge	2.2	$5 + \frac{1}{2} t2$	$\mu\text{s}$
t2	SCLK period	0.02	1000	$\mu\text{s}$
t3	SCLK setup time	2.0	$5 + \frac{1}{2} t2$	$\mu\text{s}$
t4	Last SCLK edge to CS(0:7) deassertion	2.2	$5 + \frac{1}{2} t2$	$\mu\text{s}$

## NI-845x SPI Scripting Programming Model

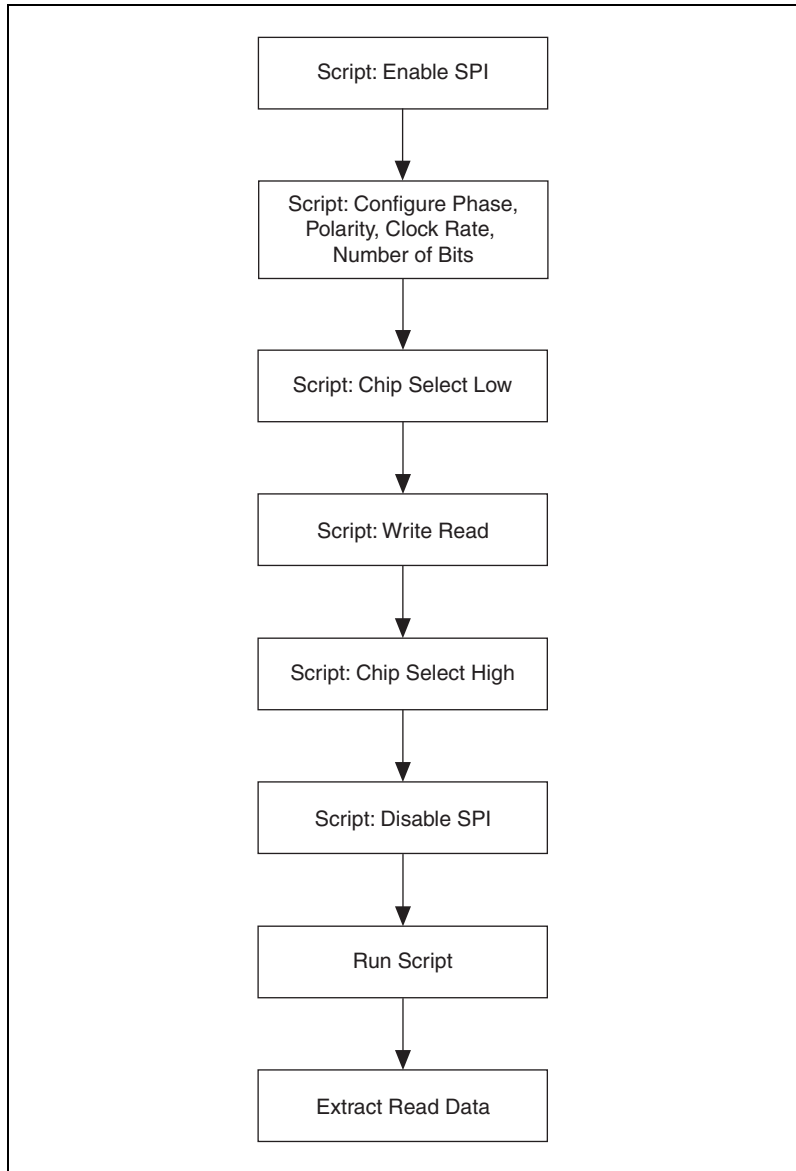
---

The SPI Scripting 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 845x 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 Scripting 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 11-3 describes an example of programming with the scripting functions for the NI-845x SPI Scripting API. The diagram is followed by a description of each step in the model.



**Figure 11-3.** 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, Number of Bits

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.

Use **NI-845x SPI Script Num Bits Per Sample.vi** in LabVIEW and `ni845xSpiScriptNumBitsPerSample` in other languages to add an SPI Script Number of Bits per Sample command to the SPI script. This command sets the number of bits per sample 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.

---

## NI-845x SPI API for LabVIEW

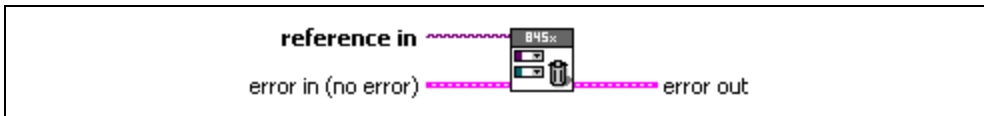
This chapter lists the LabVIEW VIs for the NI-845x SPI API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

# General Device

## NI-845x Close Reference.vi

### Purpose

Closes a previously opened reference.



### Inputs



**reference in** is a reference to an NI 845x device, I<sup>2</sup>C configuration, I<sup>2</sup>C Slave configuration, SPI configuration, SPI stream configuration, I<sup>2</sup>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.

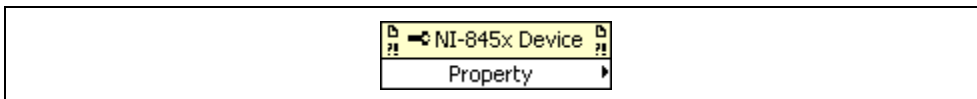


## NI-845x Device Property Node

---

### Purpose

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



### Inputs



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



### 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<sup>2</sup>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<sup>2</sup>C Pullup Enable

The **I<sup>2</sup>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.



### Timeout (ms)

The **Timeout (ms)** property sets the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.



**Note** It is highly recommended to set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

The default of this property is 30000 (30 seconds).

## NI-845x Device Reference

---

### Purpose

Specifies the device resource to be used for communication.



### Description

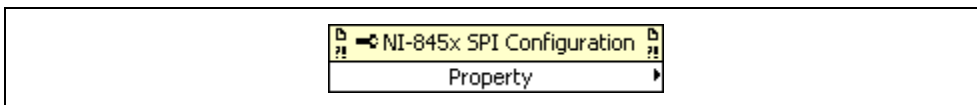
Use the **NI-845x Device Reference** to describe the NI 845x device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845x API call to invoke the function on the associated NI 845x device.

# Configuration

## NI-845x SPI Configuration Property Node

### Purpose

A property node with the NI-845x SPI Configuration class preselected. This property node allows you to query and modify SPI configuration properties of your NI 845x 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 845x 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 845x 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 configuration does not validate the clock rate until it is committed to hardware.

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).



### Number of Bits Per Sample

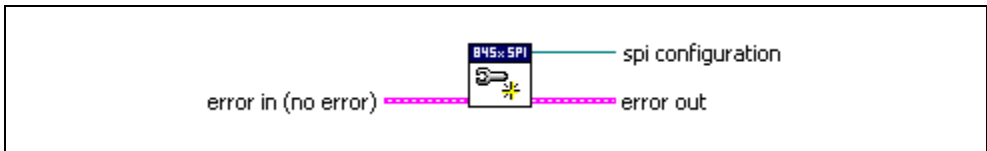
Sets the number of bits per to be clocked per sample. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), for valid settings for this property.

The default value for this property is 8.

## NI-845x SPI Create Configuration Reference.vi

### Purpose

Creates a new NI-845x 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-845x 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

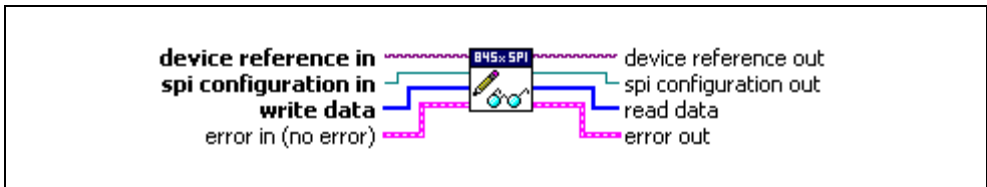
Use **NI-845x SPI Create Configuration Reference.vi** to create a new configuration to use with the NI-845x SPI Basic API. Pass the reference to a property node to make the configuration match the settings of your SPI slave. Then, pass the configuration to the SPI basic functions to execute them on the 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 845x 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 845x 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**.



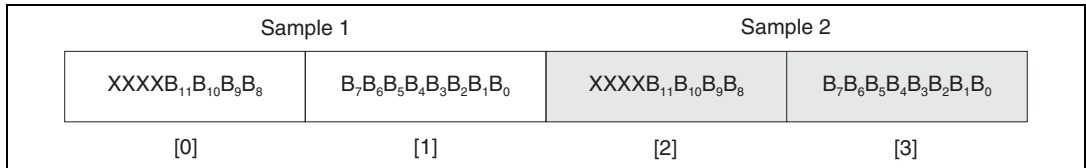
**source** identifies the VI where the error occurred.

## 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.

Data provided in **write data** and returned in **read data** is organized in big endian format. Each sample is stored using the minimum number of bytes required, with the most significant bits padded with 0.

For example, if **spi configuration in** is using 12 as the **Number of Bits Per Sample**, each sample requires 2 bytes, and the 4 most significant bits are ignored.



Any partial samples are discarded, and a warning is generated.

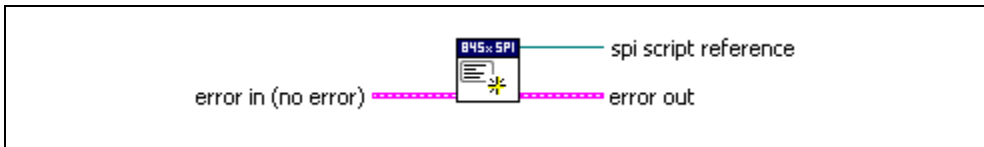
Before using **NI-845x SPI Write Read.vi**, you need to ensure that the configuration parameters specified in **spi configuration in** are correct for the device you currently want to access.

# Scripting

## NI-845x SPI Create Script Reference.vi

### Purpose

Creates a new NI-845x 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-845x 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.

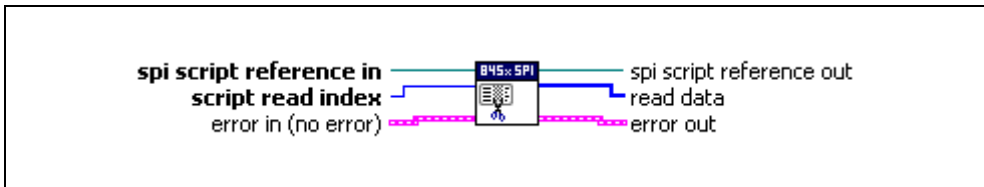
## Description

Use **NI-845x SPI Create Script Reference.vi** to create a new script to use with the NI-845x SPI Scripting 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 845x 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 845x 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.

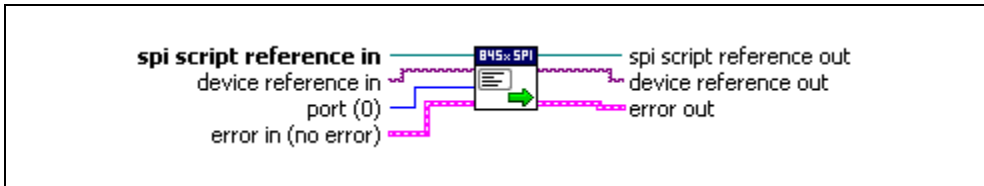
Data may be extracted for each script read in different ways. For example, you can wire the script read index output of each script read VI to its own **NI-845x SPI Extract Script Read Data.vi**. You can also place **NI-845x SPI Extract Script Read Data.vi** in a For Loop and wire the loop iteration 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 845x device.



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

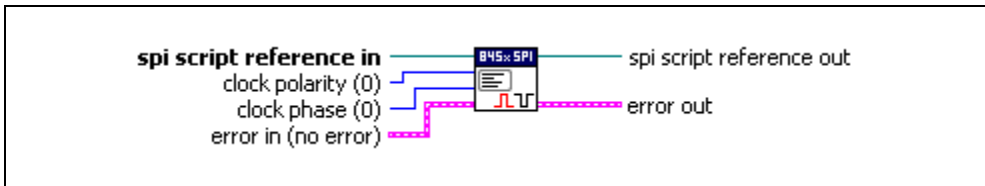
If you have multiple NI 845x devices installed in your system, you can select which device to write your SPI script to by wiring its device reference to **device reference in**. If your NI 845x device supports multiple SPI ports, you can also select which port to write your SPI script to. For single SPI port NI 845x devices, you must use the default port (0). In this way, you can create one script to run on various NI 845x devices, on various SPI ports within those devices.

**NI-845x SPI Run Script.vi** loads and executes your SPI script on the NI 845x 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 845x 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.

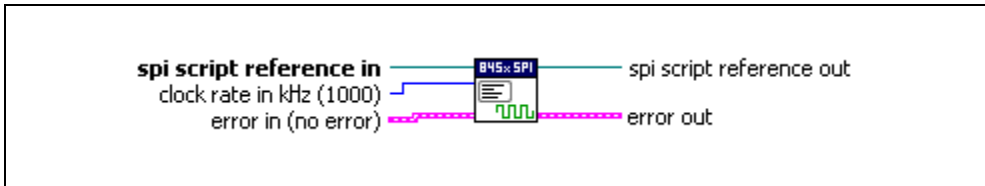
**Clock polarity** sets the idle state of the SPI clock line. The default (0) sets the clock line to idle at a low logic level. Setting the clock polarity to 1 sets the clock line to idle at a high logic level. **Clock phase** sets the SPI clock edge on which the NI 845x SPI port centers each MOSI data 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 845x device.



**clock rate in kHz** specifies the SPI clock rate. Refer to Chapter 3, *NI USB-845x Hardware Overview*, which clock rates your NI 845x 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.

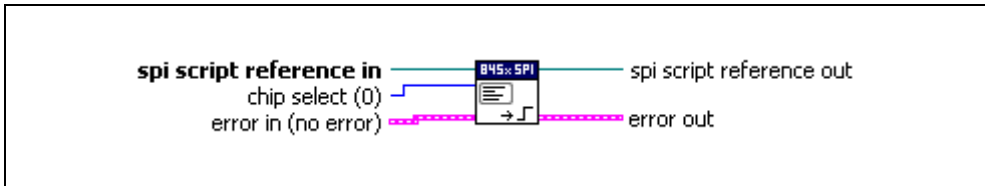
## Description

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 845x device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845x 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 845x 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

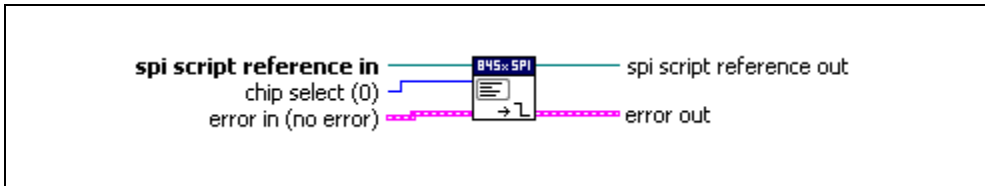
Use **NI-845x SPI Script CS High.vi** to add an SPI Script CS High command to an SPI script referenced by **spi script reference in**. This command sets an SPI chip select to the logic high state.



# 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 845x 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.

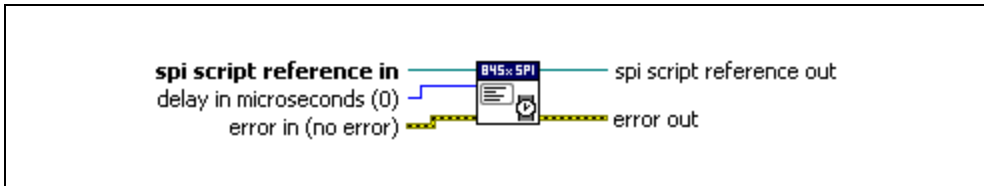
## Description

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 (Microsecond).vi

### Purpose

Adds an SPI Script Delay command to an SPI script referenced by **spi script reference in**. This command adds a microsecond 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 845x device.



**delay in microseconds** 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.

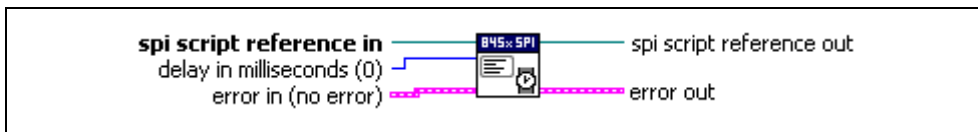
## Description

Use **NI-845x SPI Script Delay.vi** to add an SPI Script  $\mu$ s Delay command to an SPI script referenced by **spi script reference in**. This command adds a microsecond delay after the previous SPI script command.

## NI-845x SPI Script Delay (Millisecond).vi

### Purpose

Adds an SPI Script Delay command to an SPI script referenced by **spi script reference in**. This command adds a millisecond 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 845x 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.

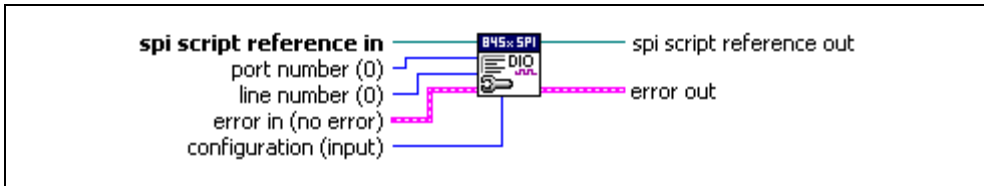
## Description

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 millisecond 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 845x device.



### Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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

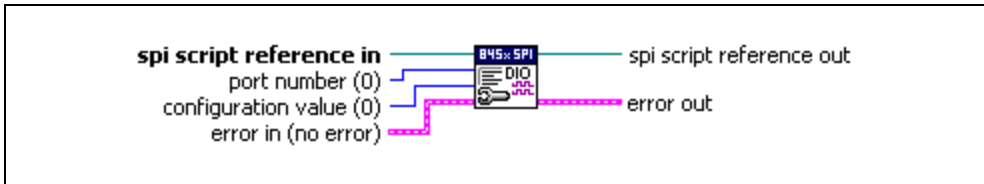
Use **NI-845x SPI Script DIO Configure Line.vi** to add an SPI Script DIO Configure Line command to an SPI script referenced by **spi script reference in**. This command allows you to configure one line, specified by **line number**, of a byte-wide DIO port, as in input or output. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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.

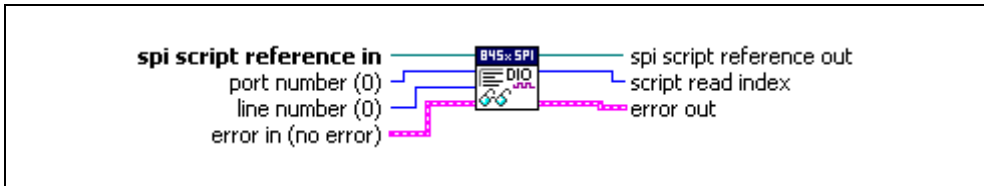
## Description

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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



## Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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

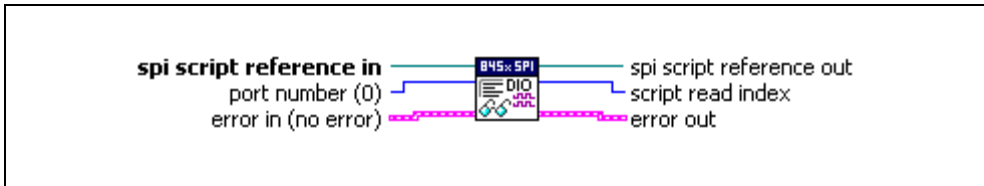
Use **NI-845x SPI Script DIO Read Line.vi** to add an SPI Script DIO Read command to an SPI script referenced by **spi script reference in**. This command allows you to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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

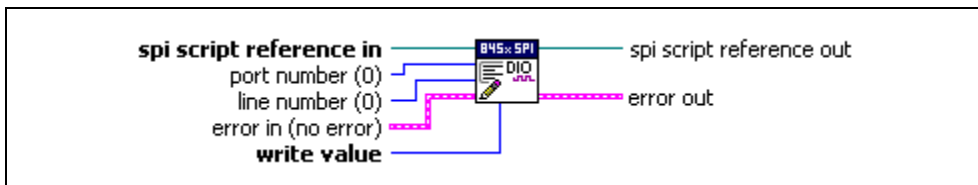
Use **NI-845x SPI Script DIO Read Port.vi** to add an SPI Script DIO Read Port command to an SPI script referenced by **spi script reference in**. This command allows you to read all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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

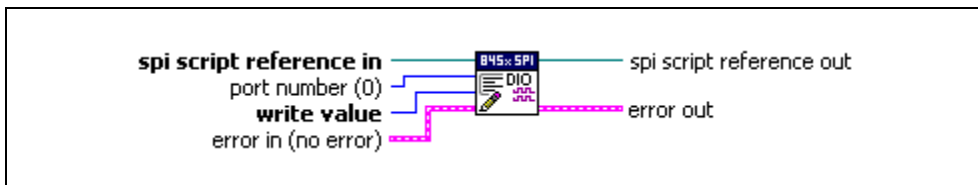
Use **NI-845x SPI Script DIO Write Line.vi** to add an SPI Script DIO Write command to an SPI script referenced by **spi script reference in**. This command allows you to write one line, specified by **line number**, of a byte-wide DIO port. If **write value** is 1, the specified line's output is driven to a high logic level. If **write value** is 0, the specified line's output is driven to a low logic level. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



**spi script reference in** is a reference to an SPI script that is run on an NI 845x 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.

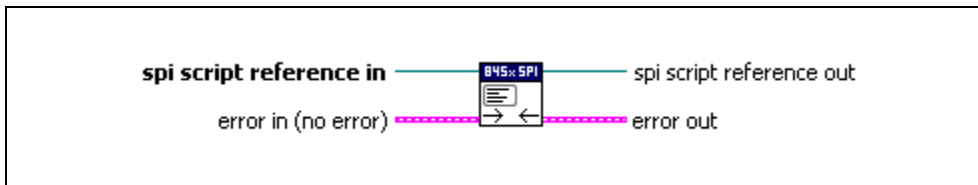
## Description

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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x 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.

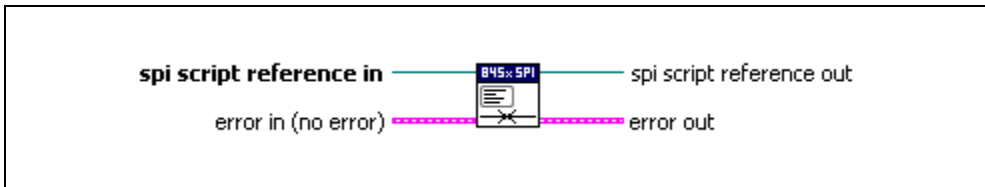
## Description

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 845x 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

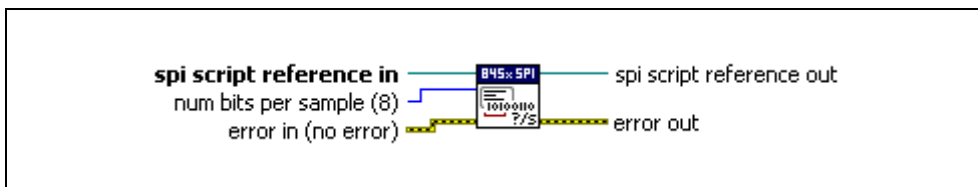
Use **NI-845x SPI Script Enable SPI.vi** to add an SPI Script Enable SPI command to an SPI script referenced by **spi script reference in**. This command switches the pins on the SPI port you specify when you use **NI-845x SPI Run Script.vi**, from tristate to master mode function.

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 Num Bits Per Sample.vi

### Purpose

Adds an SPI Script Number of Bits per Sample command to an SPI script referenced by **spi script reference in**. This command sets the number of bits per sample on an SPI port specified using **NI-845x SPI Run Script.vi**.



### Inputs



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



**num bits per sample** contains the number of bits per sample to be clocked each SPI transmission. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid settings for this property.



**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 Num Bits Per Sample.vi** to add an SPI Script Number of Bits per Sample command to an SPI script referenced by **spi script reference in**. This command sets the number of bits per sample on an SPI port specified using **NI-845x SPI Run Script.vi**.

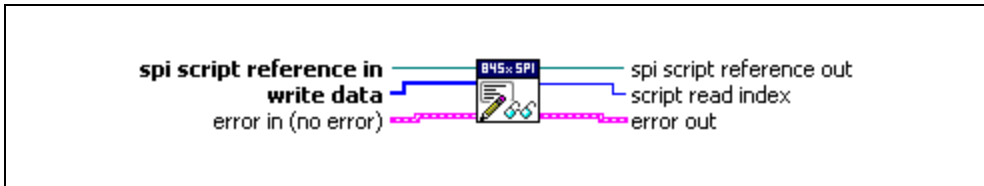
This setting remains active until the script is complete or the setting is changed.



## 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 845x 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

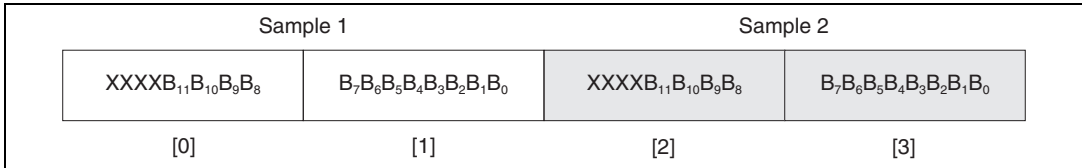
Use **NI-845x SPI Script Write Read.vi** to add an SPI Script Write Read command to an SPI script referenced by **spi script reference in**. This command exchanges an array of data with an SPI slave device connected to the SPI port you specify when you use **NI-845x SPI Run Script.vi** to execute the 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.

Data provided in **write data** and read back using **script read index** is organized in big endian format. Each sample is stored using the minimum number of bytes required, with the most significant bits padded with 0.

For example, if the **Number of Bits Per Sample** is set to 12, each sample requires 2 bytes, and the 4 most significant bits are ignored.



Any partial samples are discarded and a warning is generated.

---

# NI-845x SPI API for C

This chapter lists the functions for the NI-845x 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 scripting.

## Section Headings

---

The NI-845x 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-845x SPI API for C functions use the following data types.

<b>Data Type</b>	<b>Purpose</b>
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')
NiHandle	Operating system independent handle

## List of Functions

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

Function	Purpose
<a href="#">ni845xClose</a>	Closes a previously opened NI 845x device.
<a href="#">ni845xCloseFindDeviceHandle</a>	Closes the handles created by <a href="#">ni845xFindDevice</a> .
<a href="#">ni845xDeviceLock</a>	Locks NI 845x devices for access by a single thread.
<a href="#">ni845xDeviceUnlock</a>	Unlocks NI 845x devices.
<a href="#">ni845xFindDevice</a>	Finds an NI 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using <a href="#">ni845xFindDeviceNext</a> .
<a href="#">ni845xFindDeviceNext</a>	Finds subsequent devices after <a href="#">ni845xFindDevice</a> has been called.

Function	Purpose
<code>ni845xOpen</code>	Opens an NI 845x device for use with various write, read, and device property functions.
<code>ni845xSetIoVoltageLevel</code>	Sets the voltage level of the NI-845x I/O pins (DIO/SPI/VioRef).
<code>ni845xSpiConfigurationClose</code>	Closes a previously opened configuration.
<code>ni845xSpiConfigurationGetChipSelect</code>	Retrieves the configuration chip select value.
<code>ni845xSpiConfigurationGetClockPhase</code>	Retrieves the configuration clock phase.
<code>ni845xSpiConfigurationGetClockPolarity</code>	Retrieves the configuration clock polarity.
<code>ni845xSpiConfigurationGetClockRate</code>	Retrieves the configuration clock rate in kilohertz.
<code>ni845xSpiConfigurationGetNumBitsPerSample</code>	Retrieves the configuration number of bits per sample.
<code>ni845xSpiConfigurationGetPort</code>	Retrieves the configuration port value.
<code>ni845xSpiConfigurationOpen</code>	Creates a new NI-845x SPI configuration.
<code>ni845xSpiConfigurationSetChipSelect</code>	Sets the configuration chip select.
<code>ni845xSpiConfigurationSetClockPhase</code>	Sets the configuration clock phase.
<code>ni845xSpiConfigurationSetClockPolarity</code>	Sets the configuration clock polarity.
<code>ni845xSpiConfigurationSetClockRate</code>	Sets the configuration clock rate in kilohertz.
<code>ni845xSpiConfigurationSetNumBitsPerSample</code>	Sets the configuration number of bits per sample.
<code>ni845xSpiConfigurationSetPort</code>	Sets the configuration port number.

Function	Purpose
<code>ni845xSpiScriptClockPolarityPhase</code>	Adds an SPI Script Clock Polarity Phase command to an SPI script referenced by <code>ScriptHandle</code> . This command sets the SPI clock idle state (CPOL) and clock edge position within each data bit (CPHA).
<code>ni845xSpiScriptClockRate</code>	Adds an SPI Script Clock Rate command to an SPI script referenced by <code>ScriptHandle</code> . This command sets the SPI clock rate in kilohertz.
<code>ni845xSpiScriptClose</code>	Closes a previously opened script handle.
<code>ni845xSpiScriptCSHigh</code>	Adds an SPI Script CS High command to an SPI script referenced by <code>ScriptHandle</code> . This command sets an SPI chip select to the logic high state.
<code>ni845xSpiScriptCSLow</code>	Adds an SPI Script CS Low command to an SPI script referenced by <code>ScriptHandle</code> . This command sets an SPI chip select to the logic low state.
<code>ni845xSpiScriptDelay</code>	Adds an SPI Script Delay command to an SPI script referenced by <code>ScriptHandle</code> . This command adds a millisecond delay after the previous SPI script command.
<code>ni845xSpiScriptDioConfigureLine</code>	Adds an SPI Script DIO Configure Line command to an SPI script referenced by <code>ScriptHandle</code> . This command configures a DIO line on an NI 845x device.
<code>ni845xSpiScriptDioConfigurePort</code>	Adds an SPI Script DIO Configure Port command to an SPI script referenced by <code>ScriptHandle</code> . This command configures a DIO port on an NI 845x device.

Function	Purpose
<a href="#">ni845xSpiScriptDioReadLine</a>	Adds an SPI Script DIO Read Line command to an SPI script referenced by <code>ScriptHandle</code> . This command reads from a DIO line on an NI 845x device.
<a href="#">ni845xSpiScriptDioReadPort</a>	Adds an SPI Script DIO Read Port command to an SPI script referenced by <code>ScriptHandle</code> . This command reads from a DIO port on an NI 845x device.
<a href="#">ni845xSpiScriptDioWriteLine</a>	Adds an SPI Script DIO Write Line command to an SPI script referenced by <code>ScriptHandle</code> . This command writes to a DIO line on an NI 845x device.
<a href="#">ni845xSpiScriptDioWritePort</a>	Adds an SPI Script DIO Write Port command to an SPI script referenced by <code>ScriptHandle</code> . This command writes to a DIO port on an NI 845x device.
<a href="#">ni845xSpiScriptDisableSPI</a>	Adds an SPI Script Disable SPI command to an SPI script referenced by <code>ScriptHandle</code> . This command tristates the pins on an SPI port specified using <a href="#">ni845xSpiScriptRun</a> . It also tristates all chip select pins.
<a href="#">ni845xSpiScriptEnableSPI</a>	Adds an SPI Script Enable SPI command to an SPI script referenced by <code>ScriptHandle</code> . This command switches the pins on an SPI port specified using <a href="#">ni845xSpiScriptRun</a> to master mode function. All chip select pins are switched from tristate to push-pull output driven high.



Function	Purpose
<a href="#">ni845xSpiScriptExtractReadData</a>	Extracts the desired read data from an SPI script, referenced by <code>ScriptHandle</code> , which has been processed by <a href="#">ni845xSpiScriptRun</a> . Each script read command ( <a href="#">ni845xSpiScriptWriteRead</a> , <a href="#">ni845xSpiScriptDioReadPort</a> , <a href="#">ni845xSpiScriptDioReadLine</a> ) 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 <a href="#">ni845xSpiScriptExtractReadData</a> .
<a href="#">ni845xSpiScriptExtractReadDataSize</a>	Retrieves the read data size from an SPI script, referenced by <code>ScriptHandle</code> , which has been processed by <a href="#">ni845xSpiScriptRun</a> . Each script read command ( <a href="#">ni845xSpiScriptWriteRead</a> , <a href="#">ni845xSpiScriptDioReadPort</a> , <a href="#">ni845xSpiScriptDioReadLine</a> ) returns a script read index. You can extract data for each script read index in a script, by passing each index to <a href="#">ni845xSpiScriptExtractReadData</a> .
<a href="#">ni845xSpiScriptNumBitsPerSample</a>	Adds an SPI Number of Bits per Sample command to an SPI script referenced by <code>ScriptHandle</code> . This command sets the number of bits per sample for SPI transfers.
<a href="#">ni845xSpiScriptOpen</a>	Creates a new NI-845x SPI script.
<a href="#">ni845xSpiScriptReset</a>	Resets an SPI script referenced by <code>ScriptHandle</code> to an empty state.
<a href="#">ni845xSpiScriptRun</a>	Sends the SPI script to the desired NI 845x device, which then interprets and runs it.

Function	Purpose
<code>ni845xSpiScriptUsDelay</code>	Adds an SPI Script $\mu$ s Delay command to an SPI script referenced by <code>ScriptHandle</code> . This command adds a microsecond delay after the previous SPI script command.
<code>ni845xSpiScriptWriteRead</code>	Adds an SPI Script Write Read command to an SPI script referenced by <code>ScriptHandle</code> . This command exchanges an array of data with an SPI slave device.
<code>ni845xSpiWriteRead</code>	Exchanges an array of data with an SPI slave device.
<code>ni845xSetTimeout</code>	Sets the global timeout value.
<code>ni845xStatusToString</code>	Converts a status code into a descriptive string.

# General Device

---

## ni845xClose

---

### Purpose

Closes a previously opened NI 845x device.

### Format

```
int32 ni845xClose(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 (  
    NiHandle FindDeviceHandle  
);
```

### Inputs

NiHandle 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 845x devices for access by a single thread.

### Format

```
int32 ni845xDeviceLock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x devices.

### Format

```
int32 ni845xDeviceUnlock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using [ni845xFindDeviceNext](#).

### Format

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

### Inputs

None.

### Outputs

`char * pFirstDevice`

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

`NiHandle * 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 845x 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 845x device and the number of NI 845x 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 845x 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 (
    NiHandle FindDeviceHandle,
    char *   pNextDevice
);
```

### Inputs

NiHandle FindDeviceHandle

Describes a find list. [ni845xFindDevice](#) creates this parameter.

### Outputs

char \* pNextDevice

A pointer to the string containing the next NI 845x 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 845x device for use with various write, read, and device property functions.

### Format

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

### Inputs

char \* pResourceName

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

### Outputs

NiHandle \* 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 845x 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 845x device.

### Format

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

### Inputs

NiHandle 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 845x device. The board reference voltage is used for SPI, I<sup>2</sup>C, and DIO. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), to determine the available voltage levels on your hardware.

## ni845xSetTimeout

---

### Purpose

Modifies the global timeout for operations when using an NI 845x device.

### Format

```
int32 ni845xSetTimeout (
    NiHandle DeviceHandle,
    uInt32 Timeout
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

uInt32 Timeout

The timeout value in milliseconds. The minimum timeout is 1000 ms (1 second).

### 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 `ni845xI2cSetTimeout` to set the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.

The default of this property is 30000 (30 seconds).



**Note** You should set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

## 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-845x 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-845x 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-845x function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845x use of this status.

## NI-845x 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-845x function. If an error is detected, you should close all NI-845x 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<sup>2</sup>C 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 (  
    NiHandle ConfigurationHandle  
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    uInt32 * pChipSelect  
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    int32 * pClockPhase  
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    int32 * pClockPolarity
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    uInt16 * pClockRate  
);
```

### Inputs

NiHandle 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.

## ni845xSpiConfigurationGetNumBitsPerSample

---

### Purpose

Retrieves the configuration number of bits per sample.

### Format

```
int32 ni845xSpiConfigurationGetNumBitsPerSample (  
    NiHandle ConfigurationHandle,  
    uInt16 * pNumBitsPerSample  
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xSpiConfigurationOpen](#).

### Outputs

uInt16 \* pNumBitsPerSample

A pointer to an unsigned 16-bit integer to store the number of bits per sample 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 [ni845xSpiConfigurationGetNumBitsPerSample](#) to retrieve the number of bits per sample set in ConfigurationHandle for SPI transfers.

## ni845xSpiConfigurationGetPort

---

### Purpose

Retrieves the configuration port value.

### Format

```
int32 ni845xSpiConfigurationGetPort (  
    NiHandle ConfigurationHandle,  
    uInt8 * pPort  
);
```

### Inputs

NiHandle 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-845x SPI configuration.

### Format

```
int32 ni845xSpiConfigurationOpen (
    NiHandle * pConfigurationHandle
);
```

### Inputs

None.

### Outputs

```
NiHandle * 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-845x 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 (  
    NiHandle ConfigurationHandle,  
    uInt32 ChipSelect  
);
```

### Inputs

NiHandle 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 ni845xSpiConfigurationSetClockPhase (
    NiHandle ConfigurationHandle,
    int32 ClockPhase
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    int32 ClockPolarity
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    uInt16 ClockRate
);
```

### Inputs

NiHandle 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 845x 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 configuration does not validate the clock rate until it is committed to hardware.

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.

## ni845xSpiConfigurationSetNumBitsPerSample

---

### Purpose

Sets the configuration number of bits per sample.

### Format

```
int32 ni845xSpiConfigurationSetNumBitsPerSample (
    NiHandle ConfigurationHandle,
    uInt16 NumBitsPerSample
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xSpiConfigurationOpen](#).

uInt16 NumBitsPerSample

Specifies the number of bits per sample to be used for SPI transmissions.

The default value for the number of bits per sample is 8.

Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid settings for this property.

### 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 `ni845xSpiConfigurationSetNumBitsPerSample` to set the number of bits per sample for an SPI transmission.

## ni845xSpiConfigurationSetPort

---

### Purpose

Sets the configuration port number.

### Format

```
int32 ni845xSpiConfigurationSetPort (
    NiHandle ConfigurationHandle,
    uInt8 Port
);
```

### Inputs

NiHandle 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 845x 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 (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uInt32 WriteSize,
    uInt8 * pWriteData,
    uInt32 * pReadSize,
    uInt8 * pReadData
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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](#).

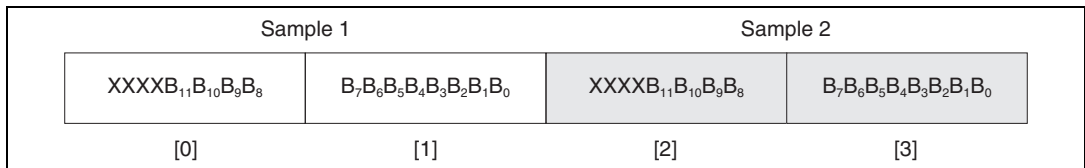
## Description

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.

Data provided in `pWriteData` and retrieved using `pReadData` is organized in big endian format. Each sample is stored using the minimum number of bytes required with the most significant bits padded with 0.

For example, if the number of bits per sample is set to 12, each sample requires 2 bytes and the 4 most significant bits are ignored.



Any partial samples are discarded, and a warning is generated.

# Scripting

---

## 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 (
    NiHandle  ScriptHandle,
    int32     Polarity,
    int32     Phase
);
```

### Inputs

`NiHandle 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-845x 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 (
    NiHandle  ScriptHandle,
    uInt16    ClockRate
);
```

### Inputs

`NiHandle 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 845x 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 845x device can clock data only at specific rates. If the selected rate is not one of the rates your hardware supports, the NI-845x 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 (NiHandle ScriptHandle);
```

### Inputs

NiHandle 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 (
    NiHandle ScriptHandle,
    uInt32   ChipSelectNum
);
```

### Inputs

`NiHandle 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 (
    NiHandle ScriptHandle,
    uInt32   ChipSelectNum
);
```

### Inputs

`NiHandle 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 millisecond delay after the previous SPI script command.

### Format

```
int32 ni845xSpiScriptDelay (  
    NiHandle ScriptHandle,  
    uInt8 Delay  
);
```

### Inputs

`NiHandle 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 millisecond 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 845x device.

### Format

```
int32 ni845xSpiScriptDioConfigureLine (
    NiHandle ScriptHandle,
    uInt8    PortNumber,
    uInt8    LineNumber,
    int32    ConfigurationValue
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 845x device.

### Format

```
int32 ni845xSpiScriptDioConfigurePort (
    NiHandle  ScriptHandle,
    uInt8     PortNumber,
    uInt8     ConfigurationValue
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the port to configure. For NI 845x 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 845x device.

### Format

```
int32 ni845xSpiScriptDioReadLine(
    NiHandle  ScriptHandle,
    uInt8     PortNumber,
    uInt8     LineNumber,
    uInt32 *  pScriptReadIndex
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the



PortNumber input to select the desired port. For NI 845x 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 845x device.

### Format

```
int32 ni845xSpiScriptDioReadPort (
    NiHandle  ScriptHandle,
    uInt8     PortNumber,
    uInt32 *  pScriptReadIndex
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 845x device.

### Format

```
int32 ni845xSpiScriptDioWriteLine (
    NiHandle ScriptHandle,
    uInt8     PortNumber,
    uInt8     LineNumber,
    int32     WriteData
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 845x device.

### Format

```
int32 ni845xSpiScriptDioWritePort (
    NiHandle ScriptHandle,
    uInt8 PortNumber,
    uInt8 WriteData
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 (  
    NiHandle ScriptHandle  
);
```

### Inputs

`NiHandle 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 (  
    NiHandle ScriptHandle  
);
```

### Inputs

`NiHandle 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 (
    NiHandle ScriptHandle,
    uInt32   ScriptReadIndex,
    uInt8 *  pReadData
);
```

### Inputs

`NiHandle 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 (
    NiHandle  ScriptHandle,
    uInt32    ScriptReadIndex,
    uInt32 *  pReadDataSize
);
```

### Inputs

`NiHandle 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.

## ni845xSpiScriptNumBitsPerSample

---

### Purpose

Adds an SPI Script Number of Bits per Sample command to an SPI script referenced by `pScriptHandle`. This command sets the number of bits per sample on an SPI port specified with [ni845xSpiScriptRun](#).

### Format

```
int32 ni845xSpiScriptNumBitsPerSample (
    NiHandle pScriptHandle,
    uInt16   NumBitsPerSample
);
```

### Inputs

`NiHandle pScriptHandle`

The script handle returned from [ni845xSpiScriptOpen](#).

`uInt16 NumBitsPerSample`

Sets the number of bits per sample to be clocked each SPI transmission. Refer to Appendix A, *NI USB-845x Hardware Specifications*, for valid settings for this property.

### Outputs

None

### 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 [ni845xSpiScriptNumBitsPerSample](#) to add an SPI Script Number of Bits per Sample command to an SPI script referenced by `pScriptHandle`. This command sets the number of bits per sample on an SPI port specified using [ni845xSpiScriptRun](#).

This setting remains active until the script is complete or the setting is changed.

# ni845xSpiScriptOpen

---

## Purpose

Creates a new NI-845x SPI script.

## Format

```
int32 ni845xSpiScriptOpen (NiHandle * pScriptHandle);
```

## Inputs

None.

## Outputs

```
NiHandle * 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-845x SPI Scripting API. Pass the reference to SPI script functions to create the script. Then, call [ni845xSpiScriptRun](#) to execute your script on your NI 845x 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 (NiHandle ScriptHandle);
```

### Inputs

`NiHandle 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 845x device, which then interprets and runs it.

### Format

```
int32 ni845xSpiScriptRun (
    NiHandle  ScriptHandle,
    NiHandle  DeviceHandle,
    uInt8     PortNumber
);
```

### Inputs

NiHandle ScriptHandle

The script handle returned from [ni845xSpiScriptOpen](#).

NiHandle 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 845x 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 845x device supports multiple SPI ports, you can also select which port to write your SPI script to. For single SPI port NI 845x devices, you must use the default port (0). In this way, you can create one script to run on various NI 845x devices, on various SPI ports within those devices. `ni845xSpiScriptRun` loads and executes your SPI script on the NI 845x 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`.

## ni845xSpiScriptUsDelay

---

### Purpose

Adds an SPI Script  $\mu$ s Delay command to an SPI script referenced by `ScriptHandle`. This command adds a microsecond delay after the previous SPI script command.

### Format

```
int32 ni845xSpiScriptUsDelay (
    NiHandle ScriptHandle,
    uInt16   Delay
);
```

### Inputs

`NiHandle ScriptHandle`

The script handle returned from [ni845xSpiScriptOpen](#).

`uInt16 Delay`

The desired delay in microseconds.

### 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 `ni845xSpiScriptUsDelay` to add an SPI Script  $\mu$ s Delay command to an SPI script referenced by `ScriptHandle`. This command adds a microsecond delay after the previous SPI script command.

## 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 (
    NiHandle  ScriptHandle,
    uInt32    WriteSize,
    uInt8 *   pWriteData,
    uInt32 *  pScriptReadIndex
);
```

### Inputs

`NiHandle 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](#).

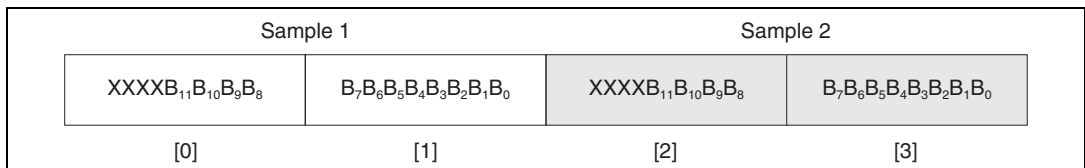
### Description

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.

Data provided in `pWriteData` and read back using `pScriptReadIndex` is organized in big endian format. Each sample is stored using the minimum number of bytes required with the most significant bits padded with 0.

For example, if the number of bits per sample is set to 12, each sample requires 2 bytes and the 4 most significant bits are ignored.



Any partial samples will be discarded and a warning will be generated.



---

# Using the NI-845x SPI Stream API

This chapter helps you get started with the NI-845x SPI Stream API.

## NI-845x 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 14-1. 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 845x 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.

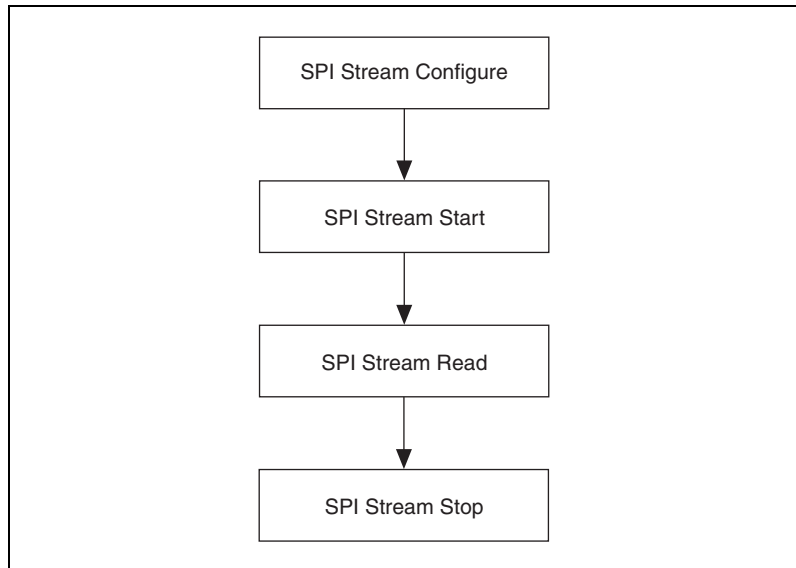


Figure 14-1. NI-845x SPI API Stream Programming Model

## 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 845x device.

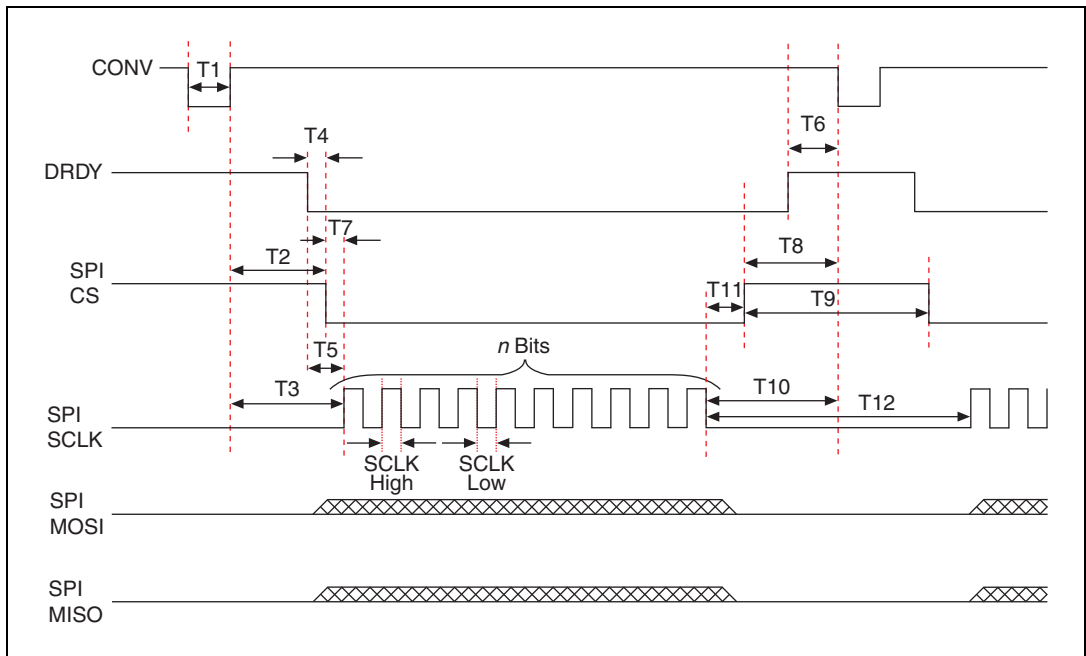
## SPI Stream Stop

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 14-2 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 14-2.** Waveform 1 Timing Diagram

Table 14-1 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 14-1. Timing Parameters

Active Pin(s) <sup>1</sup>	SCLK <sub>H</sub>	SCLK <sub>L</sub>	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>	T <sub>12</sub>
None	✓	✓	—	—	—	—	—	—	—	—	—	—	—	✓
CONV	✓	✓	✓	—	✓	—	—	—	—	—	—	✓	—	—
DRDY	✓	✓	—	—	—	—	✓	—	—	—	—	—	—	—
CS	✓	✓	—	—	—	—	—	—	✓	—	✓	—	✓	—
CONV, CS	✓	✓	✓	✓	—	—	—	—	✓	✓	—	—	✓	—
CONV, DRDY	✓	✓	✓	—	—	—	✓	✓	—	—	—	—	—	—
DRDY, CS	✓	✓	—	—	—	✓	—	—	✓	—	—	—	✓	—
CONV, DRDY, CS	✓	✓	✓	—	—	✓	—	✓	✓	—	—	—	✓	—

<sup>1</sup> 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 845x 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 845x 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 Chapter 3, *NI USB-845x Hardware Overview*, for the pinout of your NI 845x device.

---

## NI-845x SPI Stream API for LabVIEW

This chapter lists the LabVIEW VIs for the NI-845x SPI Stream API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.



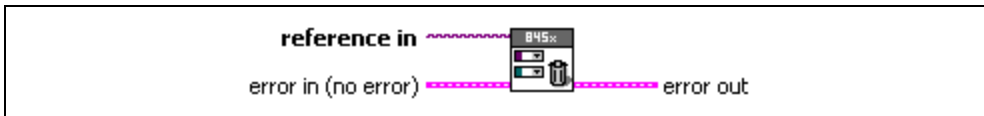
**Note** The SPI Stream API is not available for the NI USB-8451.

# General Device

## NI-845x Close Reference.vi

### Purpose

Closes a previously opened reference.



### Inputs



**reference in** is a reference to an NI 845x device, I<sup>2</sup>C configuration, I<sup>2</sup>C Slave configuration, SPI configuration, SPI stream configuration, I<sup>2</sup>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.

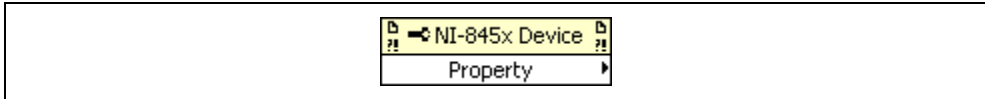
## Description

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

## NI-845x Device Property Node

### Purpose

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



### Inputs



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



### 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<sup>2</sup>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<sup>2</sup>C Pullup Enable**

The **I<sup>2</sup>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.



### **Timeout (ms)**

The **Timeout (ms)** property sets the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.



**Note** It is highly recommended to set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

The default of this property is 30000 (30 seconds).

## NI-845x Device Reference

---

### Purpose

Specifies the device resource to be used for communication.



### Description

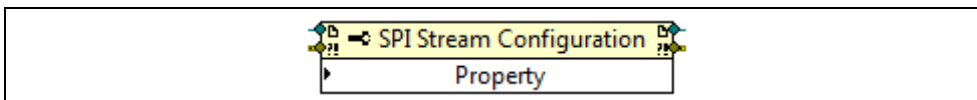
Use the **NI-845x Device Reference** to describe the NI 845x device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845x API call to invoke the function on the associated NI 845x device.

# Configuration

## NI-845x SPI Stream Configuration Property Node

### Purpose

A property node with the NI-845x 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 Appendix A, *NI USB-845x Hardware Specifications*, 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 845x device. If this property is set to 0, transfers between the host and your device occur at the standard USB packet size. For most applications, this value is optimal.

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 845x 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.

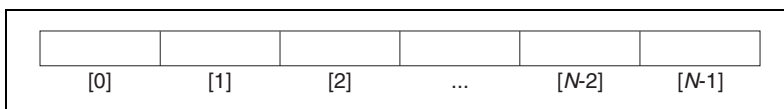
The default value for this property is 0.



### Waveform1:MOSI Data

Sets the data to be used to transfer on MOSI during an SPI operation.

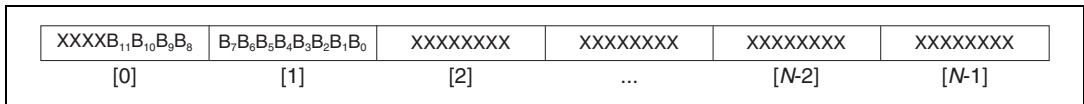
The data must be organized in big endian format. Using the array below as an example, if you were to pass in the array below, MOSI Data[N-1] contains the least-significant bits and MOSI Data[0] contains the most-significant bits.



If the total number of bits to transmit, as configured in *Number of Bits Per Sample*, is smaller than the total number of bits in the array, only the

minimum required bytes are used (starting at index 0), but they remain big endian.

For example, if the configuration has the number of bits set to 12, 12 bits are transferred. This requires 2 bytes. If you provide a data array with  $N$  elements, where  $N \geq 2$ , MOSI Data[0] contains bits 11:8 (in bits 3:0) and MOSI Data[1] contains bits 7:0. Any additional data is ignored.



If you provided MOSI Data with  $N$  elements where  $N < 2$ , the provided bytes are used as the most significant bits and additional required bytes are appended as zeros. In the example above, if you only supplied MOSI Data[0], the bit pattern on the bus would be B<sub>11</sub>B<sub>10</sub> B<sub>9</sub> B<sub>8</sub>00000000.



**Note** The NI USB-8452 uses a 100 MHz system clock. The period of each system clock is 10 ns. The SPI Stream API is not available for the NI USB-8451.



#### Waveform1:Timing:SCLKLow

Sets the number of system clocks for the SCLK low period for [Waveform 1](#).

The default value is 1.



#### Waveform1:Timing:SCLKHigh

Sets the number of system clocks for the SCLK high period for [Waveform 1](#).

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

The default value is 2.



**Note** This timing parameter may be slightly longer than specified because DRDY is an asynchronous input.



### 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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

The default value is 2.





**Note** This timing parameter may be slightly longer than specified because DRDY is an asynchronous input.



### 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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

The default value is 2.



**Note** This timing parameter may be slightly longer than specified because DRDY is an asynchronous input.



### 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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the NI-845x SPI Stream API](#), to determine the timing parameters used for your application.

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 14-2, [Waveform 1 Timing Diagram](#), in Chapter 14, [Using the](#)

*NI-845x SPI Stream API*, to determine the timing parameters used for your application.

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 14-2, *Waveform 1 Timing Diagram*, in Chapter 14, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

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 14-2, *Waveform 1 Timing Diagram*, in Chapter 14, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

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 14-2, *Waveform 1 Timing Diagram*, in Chapter 14, *Using the NI-845x SPI Stream API*, to determine the timing parameters used for your application.

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.

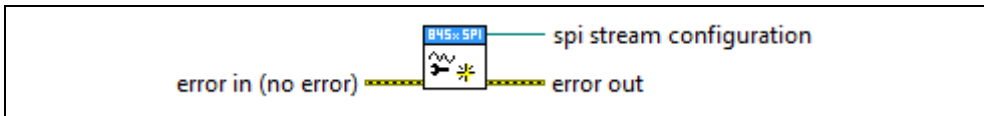
Drive Low

The pin is driven low.

## NI-845x SPI Stream Create Configuration Reference.vi

### Purpose

Creates a new NI-845x 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-845x 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

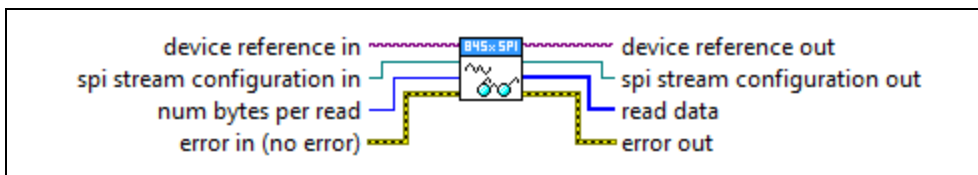
Use **NI-845x SPI Stream Create Configuration Reference.vi** to create a new configuration to use with the NI-845x SPI Stream API. Pass the reference to a property node to make the configuration match the settings of your SPI slave. Then, pass the configuration to the SPI stream functions to execute 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.

# Streaming

## NI-845x SPI Stream Read.vi

### Purpose

Reads data from an SPI slave device



### Inputs



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

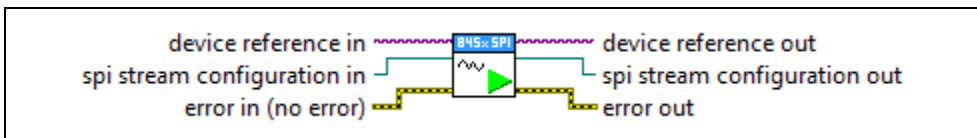


**Note** The SPI Stream API is not available for the NI USB-8451.

## NI-845x SPI Stream Start.vi

### Purpose

Starts the streaming operation on an NI 845x device.



### Inputs



**device reference in** is a reference to an NI 845x 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 845x 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 845x device into streaming mode. Once in streaming mode, your NI 845x device generates the waveform described by **spi stream configuration in**. Your NI 845x 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.

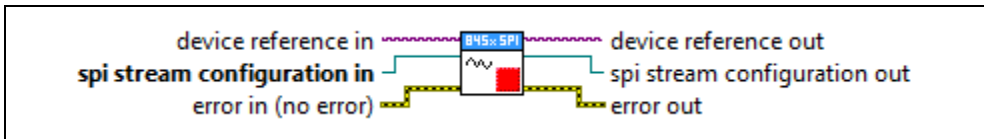


**Note** The SPI Stream API is not available for the NI USB-8451.

## NI-845x SPI Stream Stop.vi

### Purpose

Stops a streaming operation on an NI 845x device.



### Inputs



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



**Note** The SPI Stream API is not available for the NI USB-8451.

---

# NI-845x SPI Stream API for C

This chapter lists the functions for the NI-845x SPI Stream API for C and describes the format, purpose, and parameters for each function. The functions are listed alphabetically in three categories: general device, configuration, and streaming.

## Section Headings

---

The NI-845x 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-845x SPI Stream API for C functions use the following data types.

<b>Data Type</b>	<b>Purpose</b>
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')
NiHandle	Operating system independent handle

## List of Functions

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

Function	Purpose
<a href="#">ni845xClose</a>	Closes a previously opened NI 845x device.
<a href="#">ni845xCloseFindDeviceHandle</a>	Closes the handles created by <a href="#">ni845xFindDevice</a> .
<a href="#">ni845xDeviceLock</a>	Locks NI 845x devices for access by a single thread.
<a href="#">ni845xDeviceUnlock</a>	Unlocks NI 845x devices.
<a href="#">ni845xFindDevice</a>	Finds an NI 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using <a href="#">ni845xFindDeviceNext</a> .

Function	Purpose
<code>ni845xFindDeviceNext</code>	Finds subsequent devices after <code>ni845xFindDevice</code> has been called.
<code>ni845xOpen</code>	Opens an NI 845x device for use with various write, read, and device property functions.
<code>ni845xSpiStreamConfigurationClose</code>	Closes an NI-845x SPI Stream Configuration.
<code>ni845xSpiStreamConfigurationOpen</code>	Creates a new NI-845x SPI Stream Configuration.
<code>ni845xSpiStreamConfigurationGetNumBits</code>	Retrieves the configuration's number of bits per sample.
<code>ni845xSpiStreamConfigurationGetNumSamples</code>	Retrieves the configuration's number of samples to acquire.
<code>ni845xSpiStreamConfigurationGetPacketSize</code>	Retrieves the configuration's packet size.
<code>ni845xSpiStreamConfigurationGetClockPhase</code>	Retrieves the configuration's clock phase.
<code>ni845xSpiStreamConfigurationWave1GetPinConfig</code>	Retrieves the configuration's setting for an individual pin.
<code>ni845xSpiStreamConfigurationGetClockPolarity</code>	Retrieves the configuration's clock polarity.
<code>ni845xSpiStreamConfigurationWave1GetTimingParam</code>	Retrieves the configuration's setting for an individual timing parameter.
<code>ni845xSpiStreamRead</code>	Reads data from the NI 845x device.
<code>ni845xSpiStreamConfigurationWave1SetMosiData</code>	Sets the configuration's data to be transferred on MOSI.
<code>ni845xSpiStreamConfigurationSetNumBits</code>	Sets the configuration's number of bits to be transferred.
<code>ni845xSpiStreamConfigurationSetNumSamples</code>	Sets the configuration's number of samples to be transferred.

Function	Purpose
<code>ni845xSpiStreamConfigurationSetPacketSize</code>	Sets the configuration's packet size.
<code>ni845xSpiStreamConfigurationSetClockPhase</code>	Sets the configuration's clock phase.
<code>ni845xSpiStreamConfigurationWave1SetPinConfig</code>	Sets the configuration's setting for an individual pin.
<code>ni845xSpiStreamConfigurationSetClockPolarity</code>	Sets the configuration's clock polarity.
<code>ni845xSpiStreamConfigurationWave1SetTimingParam</code>	Sets the configuration's setting for an individual timing parameter.
<code>ni845xSpiStreamStart</code>	Starts the streaming operation.
<code>ni845xSpiStreamStop</code>	Stops the streaming operation.
<code>ni845xSetTimeout</code>	Sets the global timeout value.
<code>ni845xStatusToString</code>	Converts a status code into a descriptive string.

# General Device

---

## ni845xClose

---

### Purpose

Closes a previously opened NI 845x device.

### Format

```
int32 ni845xClose(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 (  
    NiHandle FindDeviceHandle  
);
```

### Inputs

NiHandle 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 845x devices for access by a single thread.

### Format

```
int32 ni845xDeviceLock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x devices.

### Format

```
int32 ni845xDeviceUnlock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using [ni845xFindDeviceNext](#).

### Format

```
int32 ni845xFindDevice (
    char      * FirstDevice,
    NiHandle * FindDeviceHandle,
    uInt32    * NumberFound
);
```

### Inputs

None.

### Outputs

`char * FirstDevice`

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

`NiHandle * 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 845x 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 845x device and the number of NI 845x 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 845x 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 (
    NiHandle FindDeviceHandle,
    char *    NextDevice
);
```

### Inputs

NiHandle FindDeviceHandle

Describes a find list. [ni845xFindDevice](#) creates this parameter.

### Outputs

char \* NextDevice

A pointer to the string containing the next NI 845x 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 845x device for use with various write, read, and device property functions.

### Format

```
int32 ni845xOpen (  
    char * ResourceName,  
    NiHandle * DeviceHandle  
);
```

### Inputs

char \* ResourceName

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

### Outputs

NiHandle \* 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 845x 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.

## ni845xSetTimeout

---

### Purpose

Modifies the global timeout for operations when using an NI 845x device.

### Format

```
int32 ni845xSetTimeout (
    NiHandle DeviceHandle,
    uInt32 Timeout
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

uInt32 Timeout

The timeout value in milliseconds. The minimum timeout is 1000 ms (1 second).

### 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 [ni845xI2cSetTimeout](#) to set the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.

The default of this property is 30000 (30 seconds).



**Note** You should set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.



## 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-845x 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-845x 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-845x function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845x use of this status.

## NI-845x 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-845x function. If an error is detected, you should close all NI-845x 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<sup>2</sup>C 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

---

## ni845xSpiStreamConfigurationClose

---

### Purpose

Closes a previously opened SPI stream configuration.

### Format

```
int32 ni845xSpiStreamConfigurationClose (
    NiHandle ConfigurationHandle
);
```

### Inputs

NiHandle 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-845x SPI stream configuration.

### Format

```
int32 ni845xSpiStreamConfigurationOpen (  
    NiHandle * ConfigurationHandle  
);
```

### Inputs

None.

### Outputs

```
NiHandle * 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-845x 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 (  
    NiHandle ConfigurationHandle,  
    uInt8 * NumBits  
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    uInt32 * NumSamples  
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    uInt32 * PacketSize  
);
```

### Inputs

NiHandle 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 845x device.

## ni845xSpiStreamConfigurationGetClockPhase

---

### Purpose

Retrieves the configuration's clock phase.

### Format

```
int32 ni845xSpiStreamConfigurationGetClockPhase (
    NiHandle ConfigurationHandle,
    uInt8 * ClockPhase
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    uInt8 PinNumber,
    uInt8 * Mode
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    uInt8 * ClockPolarity
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    uInt8 TimingParameter,
    uInt32 * ParameterValue
);
```

### Inputs

NiHandle 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 845x device.

## ni845xSpiStreamConfigurationWave1SetMosiData

### Purpose

Sets the configuration MOSI data.

### Format

```
int32 ni845xSpiStreamConfigurationWave1SetMosiData (
    NiHandle ConfigurationHandle,
    uInt8 *   DataArray,
    uInt32   ArraySize
);
```

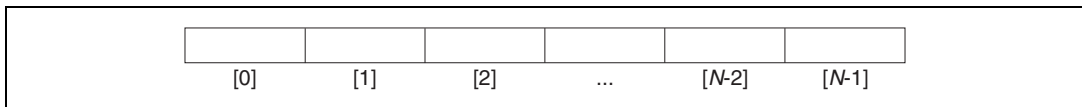
### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xSpiStreamConfigurationOpen](#).

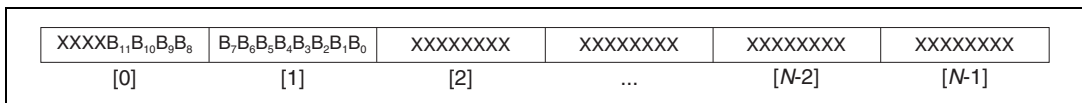
uInt8\* DataArray

An array of unsigned 8-bit integers used to specify the data transferred on MOSI. The data must be organized in big endian format. Using the array below as an example, if you were to pass in the array below, DataArray[N-1] contains the least-significant bits and DataArray[0] contains the most-significant bits.



If the total number of bits to transmit, as configured in [ni845xSpiStreamConfigurationSetNumBits](#), is smaller than the total number of bits in the array, only the minimum required bytes are used (starting at index 0), but they remain big endian.

For example, if the configuration has the number of bits set to 12, 12 bits are transferred. This requires 2 bytes. If you provide a data array with  $N$  elements, where  $N \geq 2$ , DataArray[0] contains bits 11:8 (in bits 3:0) and DataArray[1] contains bits 7:0. Any additional data is ignored.



If you provided DataArray with  $N$  elements where  $N < 2$ , the provided bytes are used as the most significant bits and additional required bytes are appended as zeros. In the example above, if you only supplied DataArray[0], the bit pattern on the bus would be B<sub>11</sub>B<sub>10</sub> B<sub>9</sub> B<sub>8</sub>00000000.

uint32\_t 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.

## ni845xSpiStreamConfigurationSetNumBits

---

### Purpose

Sets the configuration's number of bits per sample.

### Format

```
int32 ni845xSpiStreamConfigurationSetNumBits (  
    NiHandle ConfigurationHandle,  
    uInt8 NumBits  
);
```

### Inputs

NiHandle 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 (  
    NiHandle ConfigurationHandle,  
    uInt32 NumSamples  
);
```

### Inputs

NiHandle 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 845x device automatically stops streaming after the specified number of samples have been transferred.



## ni845xSpiStreamConfigurationSetPacketSize

---

### Purpose

Sets the configuration's packet size.

### Format

```
int32 ni845xSpiStreamConfigurationSetPacketSize (
    NiHandle ConfigurationHandle,
    uInt32 PacketSize
);
```

### Inputs

NiHandle 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 845x device. If this property is set to 0, transfers between the host and your device occur at the standard USB packet size. For most applications, this value is optimal.

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 845x 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.

The default value of this property is 0.

## ni845xSpiStreamConfigurationSetClockPhase

---

### Purpose

Sets the configuration's clock phase.

### Format

```
int32 ni845xSpiStreamConfigurationSetClockPhase (
    NiHandle ConfigurationHandle,
    UInt8 ClockPhase
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xSpiStreamConfigurationOpen](#).

UInt8 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 (
    NiHandle ConfigurationHandle,
    uInt8 PinNumber,
    uInt8 Mode
);
```

### Inputs

NiHandle 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 (
    NiHandle ConfigurationHandle,
    UInt8 ClockPolarity
);
```

### Inputs

NiHandle ConfigurationHandle

The configuration handle returned from [ni845xSpiStreamConfigurationOpen](#).

UInt8 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 (
    NiHandle ConfigurationHandle,
    uInt8 TimingParameter,
    uInt32 ParameterValue
);
```

### Inputs

NiHandle 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 845x device.



**Note** The NI USB-8452 uses a 100 MHz system clock. The period of each system clock is 10 ns. The SPI Stream API is not available for the NI USB-8451.



**Note** Timing parameters using DRDY may be slightly longer than specified because DRDY is an asynchronous input.

# Streaming

---

## ni845xSpiStreamRead

---

### Purpose

Reads streaming data from an NI 845x device.

### Format

```
int32 ni845xSpiStreamRead (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle,
    uint32 NumBytesToRead,
    uInt8 * ReadData,
    uInt32 * ReadSize
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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 845x 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`.



**Note** The SPI Stream API is not available for the NI USB-8451.



## ni845xSpiStreamStart

---

### Purpose

Starts the streaming operation on an NI 845x device.

### Format

```
int32 ni845xSpiStreamStart (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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 845x device into streaming mode. Once in streaming mode, your NI 845x device performs the SPI operations set to `ConfigurationHandle`. Your NI 845x 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.



**Note** The SPI Stream API is not available for the NI USB-8451.

## ni845xSpiStreamStop

---

### Purpose

Stops a streaming operation on an NI 845x device.

### Format

```
int32 ni845xSpiStreamStop (
    NiHandle DeviceHandle,
    NiHandle ConfigurationHandle
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

NiHandle 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 845x 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.



**Note** The SPI Stream API is not available for the NI USB-8451.

---

# Using the NI-845x DIO API

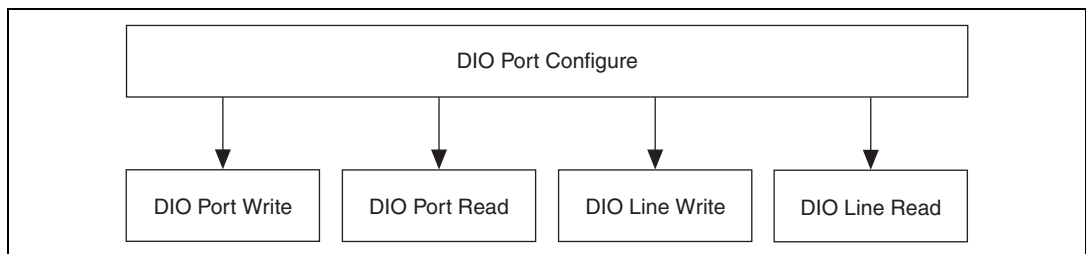
This chapter helps you get started with the DIO API.

## 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 17-1 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 17-1.** 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.
- **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<sup>2</sup>C pins are pulled-up to if using internal I<sup>2</sup>C 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.

---

## NI-845x DIO API for LabVIEW

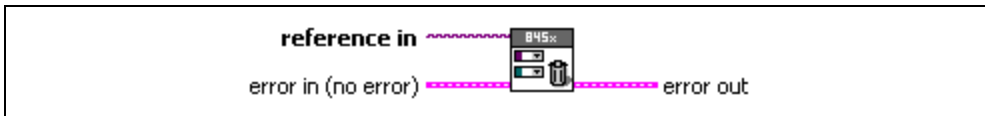
This chapter lists the LabVIEW VIs for the NI-845x DIO API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically.

# General Device

## NI-845x Close Reference.vi

### Purpose

Closes a previously opened reference.



### Inputs



**reference in** is a reference to an NI 845x device, I<sup>2</sup>C configuration, I<sup>2</sup>C Slave configuration, SPI configuration, SPI stream configuration, I<sup>2</sup>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.

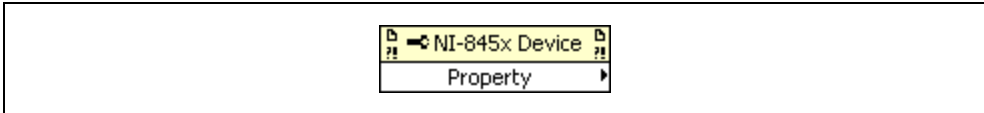
## Description

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

## NI-845x Device Property Node

### Purpose

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



### Inputs



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



### 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<sup>2</sup>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<sup>2</sup>C Pullup Enable

The **I<sup>2</sup>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.



### Timeout (ms)

The **Timeout (ms)** property sets the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.



**Note** It is highly recommended to set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.

The default of this property is 30000 (30 seconds).

## NI-845x Device Reference

---

### Purpose

Specifies the device resource to be used for communication.



### Description

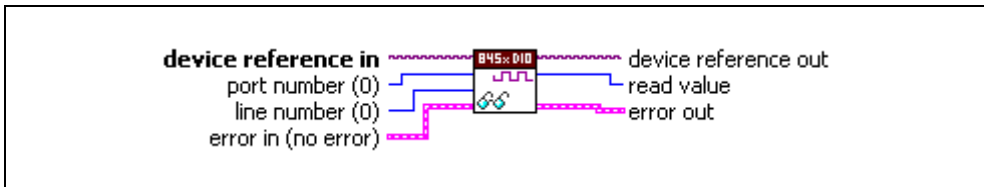
Use the NI-845x Device Reference to describe the NI 845x device to communicate with. You can wire the reference into a property node to set specific device parameters or to an NI-845x API call to invoke the function on the associated NI 845x device.

# Basic

## NI-845x DIO Read Line.vi

### Purpose

Reads from a DIO line on an NI 845x device.



### Inputs



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

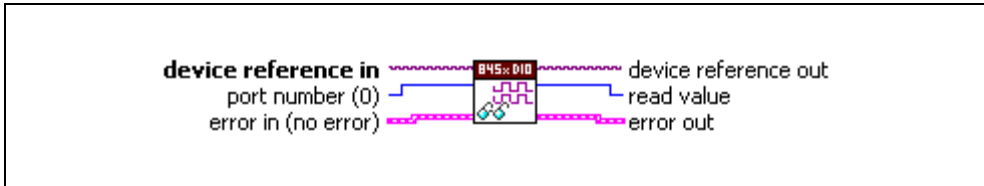
## Description

Use **NI-845x DIO Read Line.vi** to read one line, specified by **line number**, of a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



**device reference in** is a reference to an NI 845x 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 845x 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  $n = \text{DIO } n$ .



**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.

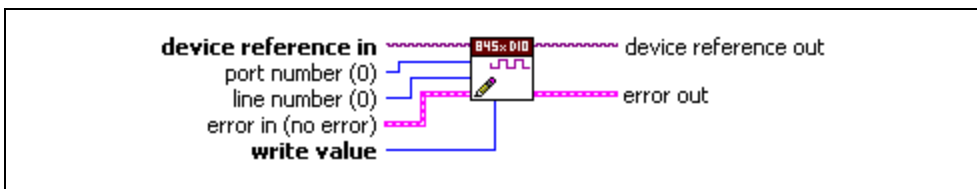
## Description

Use **NI-845x DIO Read Port.vi** to read all 8 bits on a byte-wide DIO port. For NI 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



### Inputs



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

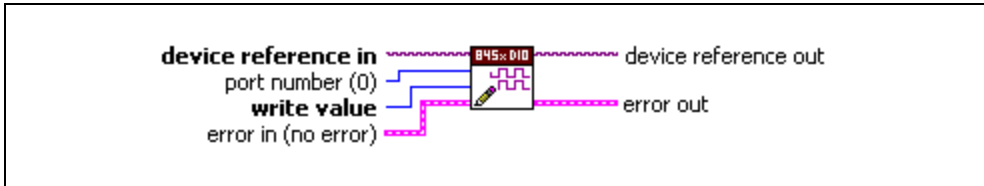
## Description

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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x 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 845x device.



## Inputs



**device reference in** is a reference to an NI 845x 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 845x 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 845x devices with multiple DIO ports, use the **port number** input to select the desired port. For NI 845x devices with one DIO port, **port number** must be left at the default (0).

---

# NI-845x DIO API for C

This chapter lists the functions for the NI-845x 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-845x 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-845x DIO API for C functions use the following data types.

<b>Data Type</b>	<b>Purpose</b>
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')
NiHandle	Operating system independent handle

## List of Functions

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

Function	Purpose
<a href="#">ni845xClose</a>	Closes a previously opened NI 845x device.
<a href="#">ni845xCloseFindDeviceHandle</a>	Closes the handles created by <a href="#">ni845xFindDevice</a> .
<a href="#">ni845xDeviceLock</a>	Locks NI 845x devices for access by a single thread.
<a href="#">ni845xDeviceUnlock</a>	Unlocks NI 845x devices.
<a href="#">ni845xDioReadLine</a>	Reads from a DIO line on an NI 845x device.
<a href="#">ni845xDioReadPort</a>	Reads from a DIO port on an NI 845x device.
<a href="#">ni845xDioSetDriverType</a>	Configures the driver type used when sourcing DIO signals on an NI 845x device.
<a href="#">ni845xDioSetPortLineDirectionMap</a>	Configures a DIO port on an NI 845x device for input or output.
<a href="#">ni845xDioWriteLine</a>	Writes to a DIO line on an NI 845x device.

Function	Purpose
<code>ni845xDioWritePort</code>	Writes to a DIO port on an NI 845x device.
<code>ni845xFindDevice</code>	Finds an NI 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using <code>ni845xFindDeviceNext</code> .
<code>ni845xFindDeviceNext</code>	Finds subsequent devices after <code>ni845xFindDevice</code> has been called.
<code>ni845xOpen</code>	Opens an NI 845x device for use with various write, read, and device property functions.
<code>ni845xSetIoVoltageLevel</code>	Sets the voltage level of the NI-845x I/O pins (DIO/SPI/VioRef).
<code>ni845xSetTimeout</code>	Sets the global timeout value.
<code>ni845xStatusToString</code>	Converts a status code into a descriptive string.

# General Device

---

## ni845xClose

---

### Purpose

Closes a previously opened NI 845x device.

### Format

```
int32 ni845xClose(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 (  
    NiHandle FindDeviceHandle  
);
```

### Inputs

NiHandle 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 845x devices for access by a single thread.

### Format

```
int32 ni845xDeviceLock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x devices.

### Format

```
int32 ni845xDeviceUnlock(NiHandle DeviceHandle);
```

### Inputs

NiHandle 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 845x 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 845x device and returns the total number of NI 845x devices present. You can find subsequent devices using [ni845xFindDeviceNext](#).

### Format

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

### Inputs

None.

### Outputs

`char * pFirstDevice`

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

`NiHandle * 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 845x 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 845x device and the number of NI 845x 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 845x 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 (
    NiHandle FindDeviceHandle,
    char *    pNextDevice
);
```

### Inputs

NiHandle FindDeviceHandle

Describes a find list. [ni845xFindDevice](#) creates this parameter.

### Outputs

char \* pNextDevice

A pointer to the string containing the next NI 845x 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 845x device for use with various write, read, and device property functions.

### Format

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

### Inputs

char \* pResourceName

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

### Outputs

NiHandle \* 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 845x 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 845x device.

### Format

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

### Inputs

NiHandle 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 845x device. The board reference voltage is used for SPI, I<sup>2</sup>C, and DIO. Refer to Appendix A, [NI USB-845x Hardware Specifications](#), to determine the available voltage levels on your hardware.

## ni845xSetTimeout

---

### Purpose

Modifies the global timeout for operations when using an NI 845x device.

### Format

```
int32 ni845xSetTimeout (
    NiHandle DeviceHandle,
    uInt32 Timeout
);
```

### Inputs

NiHandle DeviceHandle

Device handle returned from [ni845xOpen](#).

uInt32 Timeout

The timeout value in milliseconds. The minimum timeout is 1000 ms (1 second).

### 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 [ni845xI2cSetTimeout](#) to set the global timeout for the device. This timeout is the minimum amount of time an I<sup>2</sup>C, SPI, or DIO operation is allowed to complete.

The default of this property is 30000 (30 seconds).



**Note** You should set this property higher than the expected I/O time. For the NI USB-8451, a timeout may leave the device in an unknown state that may require a power cycle of the device.



## 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-845x 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-845x 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-845x function returns a status code as a signed 32-bit integer. The following table summarizes the NI-845x use of this status.

## NI-845x 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-845x function. If an error is detected, you should close all NI-845x 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<sup>2</sup>C 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 845x device.

### Format

```
int32 ni845xDioReadLine (  
    NiHandle DeviceHandle,  
    uInt8    PortNumber,  
    uInt8    LineNumber,  
    int32 *  pReadData  
);
```

### Inputs

NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x 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 845x device.

### Format

```
int32 ni845xDioReadPort (  
    NiHandle DeviceHandle,  
    uInt8 PortNumber,  
    uInt8 * pReadData  
);
```

### Inputs

NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).

## ni845xDioSetPortLineDirectionMap

---

### Purpose

Configures a DIO port on an NI 845x device for input or output.

### Format

```
int32 ni845xDioSetPortLineDirectionMap (
    NiHandle DeviceHandle,
    uInt8    DioPort,
    uInt8    Map
);
```

### Inputs

NiHandle 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 845x device for input or output.

## ni845xDioSetDriverType

---

### Purpose

Configures the driver type used when sourcing DIO signals on an NI 845x device.

### Format

```
int32 ni845xDioSetDriverType (
    NiHandle DeviceHandle,
    uInt8    DioPort,
    uInt8    Type
);
```

### Inputs

NiHandle 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 845x device uses to source DIO signals.

## ni845xDioWriteLine

---

### Purpose

Writes to a DIO line on an NI 845x device.

### Format

```
int32 ni845xDioWriteLine (
    NiHandle DeviceHandle,
    uInt8    PortNumber,
    uInt8    LineNumber,
    int32    WriteData
);
```

### Inputs

`NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).



## ni845xDioWritePort

---

### Purpose

Writes to a DIO port on an NI 845x device.

### Format

```
int32 ni845xDioWritePort (  
    NiHandle DeviceHandle,  
    uInt8    PortNumber,  
    uInt8    WriteData  
);
```

### Inputs

NiHandle 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 845x devices with multiple DIO ports, use the `PortNumber` input to select the desired port. For NI 845x devices with one DIO port, leave `PortNumber` at the default (0).

---

# NI USB-845x Hardware Specifications

This appendix lists the NI USB-845x 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
<b>Input</b>			
Input low voltage	-0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	—	50	µA
<b>Output</b>			
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 <sub>cc</sub> <sup>1</sup>	V <sub>cc</sub> <sup>1</sup>	V
<sup>1</sup> V <sub>cc</sub> 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)

Transfer size .....8 bits

## Digital logic levels

Level	Min	Max	Units
<b>Input</b>			
Input low voltage	-0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	—	50	μA
<b>Output</b>			
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<sup>2</sup>C Interface**

## Signals

SDA ..... Output/input

SCL ..... Output (250 kHz max)

## Supported clock rates

I<sup>2</sup>C Standard Mode..... 32 kHz, 40 kHz, 50 kHz, 64 kHz,  
80 kHz, 100 kHzI<sup>2</sup>C Fast Mode ..... 125 kHz, 160 kHz, 200 kHz,  
250 kHz

Fast Mode Plus..... Not supported

I<sup>2</sup>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
<b>Output</b>			
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<sup>2</sup>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  $\mu$ A standby mode,  
500  $\mu$ A 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  $\times$  8.51 cm  $\times$  2.31 cm  
(2.50 in.  $\times$  3.35 in.  $\times$  0.91 in.)

With connectors ..... 8.18 cm  $\times$  8.51 cm  $\times$  2.31 cm  
(3.22 in.  $\times$  3.35 in.  $\times$  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  $\cdot$  m  
(2.0 to 2.2 lb  $\cdot$  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.

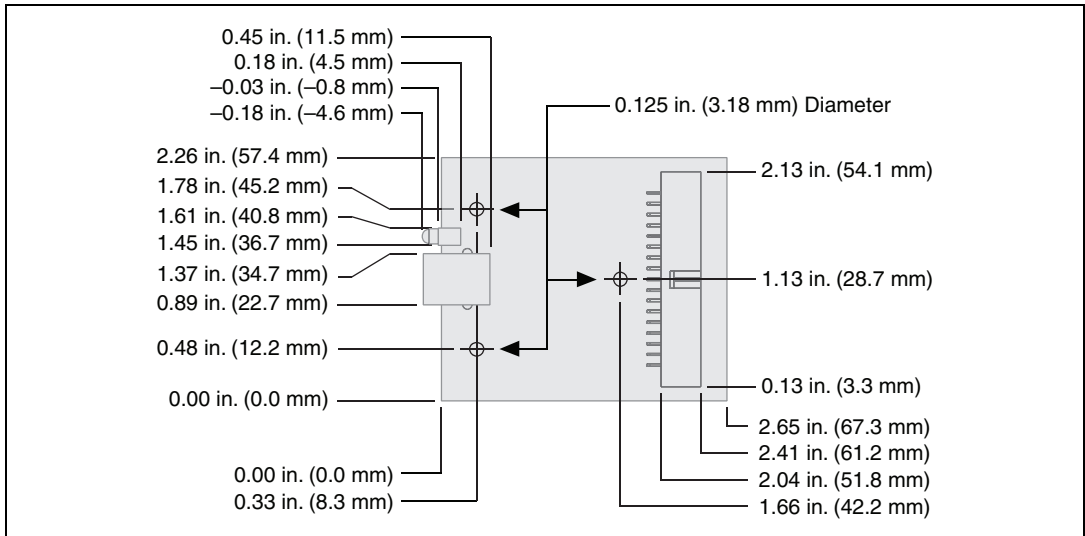


Figure A-1. USB-8451 OEM Dimensions (Top View)

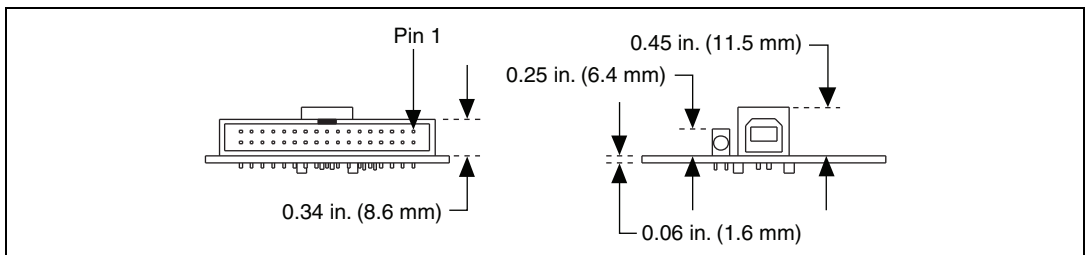


Figure A-2. USB-8451 OEM Dimensions (Front and Rear Views)

## 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) or open-drain, software selectable
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

<b>Output Specifications</b>			
<b>Logic Family</b>	<b>Voltage Low Level (V<sub>OL</sub>) (Full Temperature)</b>	<b>Voltage High Level (V<sub>OH</sub>) (Full Temperature)</b>	<b>Output Drive Strength (I<sub>O_MAX</sub>)</b>
	<b>Max (I<sub>OL</sub> = 100 <math>\mu</math>A)</b>	<b>Min (I<sub>OH</sub> = 100 <math>\mu</math>A)</b>	<b>Max</b>
1.2 V	0.2 V	1.0 V	$\pm 3$ mA
1.5 V	0.2 V	1.3 V	$\pm 6$ mA
1.8 V	0.2 V	1.6 V	$\pm 8$ mA
2.5 V	0.2 V	2.3 V	$\pm 9$ mA
3.3 V	0.2 V	3.1 V	$\pm 12$ mA
Output Impedance	70 $\Omega$ (typical)		
<b>Input Specifications</b>			
<b>Logic Family</b>	<b>Input Voltage Low (V<sub>IL</sub>) Max</b>	<b>Input Voltage High (V<sub>IH</sub>) Min</b>	
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, $\pm 50$ mA maximum		

**SPI Interface**

## Signals

SPI CS &lt;0..7&gt; ..... Output

SPI MOSI (SDO) ..... Output

SPI MISO (SDI) ..... Input

SPI CLK (SCLK) ..... Output (50 MHz max)

SPI system clock ..... 100 MHz (10 ns period)



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 k $\Omega$ ) pull down to GND
Transfer size .....	4 to 64 bits, software selectable
Bit ordering .....	Most significant bit (msb) first

## SPI specifications under different logic levels

<b>Output Specifications</b>			
<b>Logic Family</b>	<b>Voltage Low Level (V<sub>OL</sub>) (Full Temperature)</b>	<b>Voltage High Level (V<sub>OH</sub>) (Full Temperature)</b>	<b>Output Drive Strength (I<sub>O_MAX</sub>)</b>
	<b>Max (I<sub>OL</sub> = 100 uA)</b>	<b>Min (I<sub>OH</sub> = 100 uA)</b>	<b>Max</b>
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)		
<b>Input Specifications</b>			
<b>Logic Family</b>	<b>Input Voltage Low (V<sub>IL</sub>) Max</b>	<b>Input Voltage High (V<sub>IH</sub>) Min</b>	
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		

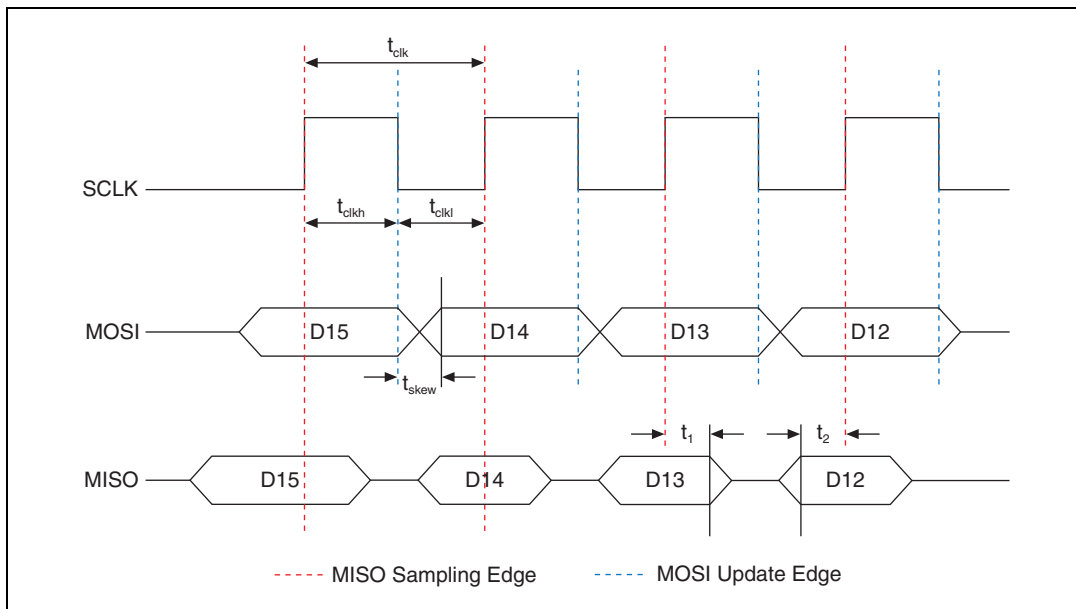
## SPI timing requirements

<b>Timing Parameter<sup>1</sup></b>	<b>Min</b>	<b>Max</b>	<b>Unit</b>
t <sub>clk</sub> SCLK period	20	—	ns
t <sub>ckl</sub> SCLK low time	9	—	ns
t <sub>ckh</sub> SCLK high time	9	—	ns
t <sub>skew</sub> MOSI output skew (with regard to SCLK edge)	-2	2	ns

Timing Parameter <sup>1</sup>	Min	Max	Unit
$t_1$ MISO hold time	5	—	ns
$t_2$ MISO setup time	4	—	ns

<sup>1</sup> All timing parameters are measured/required at IDC connector.

SPI timing diagram



## I<sup>2</sup>C Interface

### Signals

SDA .....Output/input

SCL .....Output/input

### Supported clock rates (Master Mode)

I<sup>2</sup>C Standard Mode ..... 16 kHz, 20 kHz, 25 kHz, 31 kHz,  
40 kHz, 50 kHz, 62 kHz, 80 kHz,  
100 kHz

I<sup>2</sup>C Fast Mode ..... 125 kHz, 200 kHz, 250 kHz,  
400 kHz

- I<sup>2</sup>C Fast Mode Plus ..... 500 kHz, 1 MHz
- I<sup>2</sup>C High Speed Mode ..... 1.11 MHz, 1.33 MHz, 2.22 MHz, 3.33 MHz
- Supported clock rates (Slave Mode) ..... Up to 3.4 MHz<sup>1</sup>
- 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<sup>2</sup>C I/O specifications under different logic levels

Logic Family	Output Voltage Low (V <sub>OL</sub> ) Max	Input Voltage Low (V <sub>IL</sub> ) 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) <sup>1</sup>	
Onboard capacitance	70 pF (max)	
Input protection	40 mA (max)	
<sup>1</sup> With onboard pull-up resistors enabled (tested under V <sub>OL</sub> = 0.24 V)		



**Note** This interface is compatible with both I<sup>2</sup>C and SMBus devices. (SMBus compatibility is only under V<sub>ref</sub>= 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.)

<sup>1</sup> To support Slave Mode under logic standards below 2.5 V, the master device should meet 70 ns setup time between SDA and SCL.

## 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,  
with  $\pm 10\%$  tolerance, software  
selectable

Current.....20 mA max

## Physical Characteristics

### NI USB-8452

Dimensions .....7.26 cm  $\times$  9.19 cm  $\times$  2.03 cm  
(2.86 in.  $\times$  3.62 in.  $\times$  0.8 in.)

I/O connectors.....1  $\times$  right angle USB series B  
receptacle  
1  $\times$  right angle male IDE cable  
receptacle

Weight .....79 g (2.8 oz)

## NI USB-8452 OEM

Dimensions.....	6.65 cm × 8.86 cm (2.62 in. × 3.49 in.)
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.

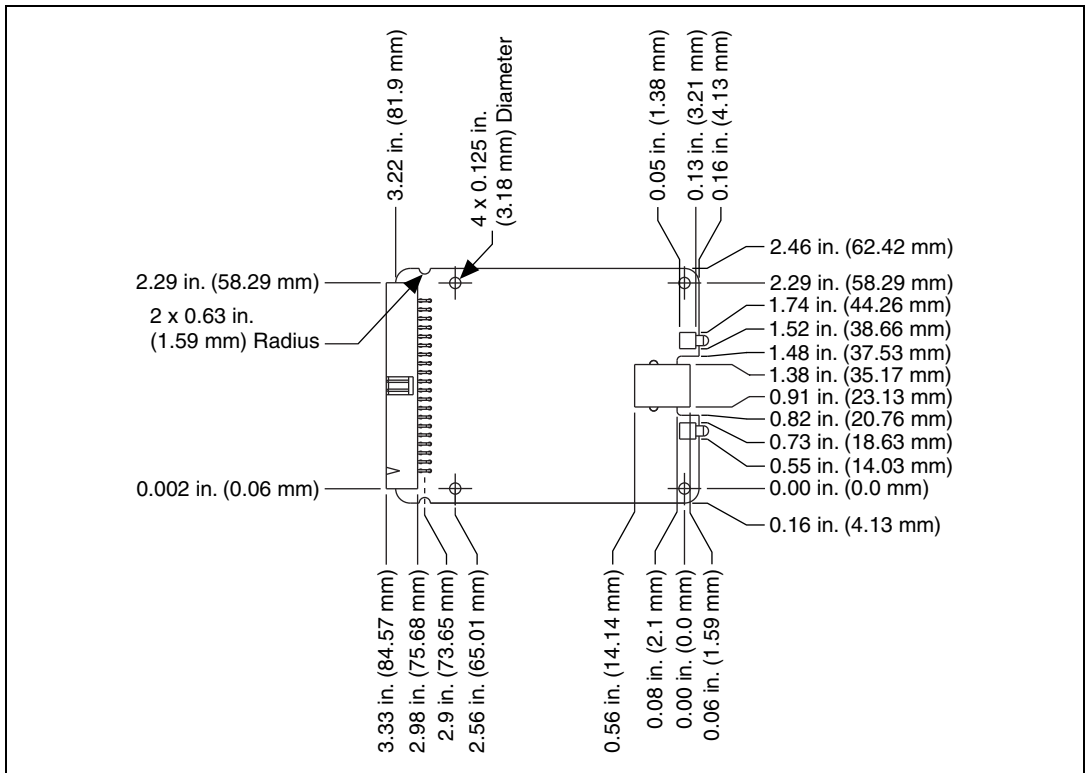
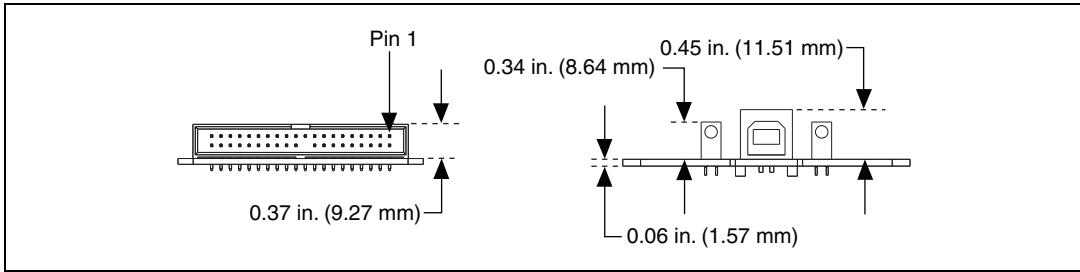


Figure A-3. USB-8452 OEM Dimensions (Top View)



**Figure A-4.** USB-8452 OEM Dimensions (Front and Rear Views)

## Safety

### Safety Standards

The NI USB-845x modules meet 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-845x 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-8452

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

## 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.



**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.

## CE Compliance

### 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-8452

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](http://ni.com/certification), search by model number or product line, and click the appropriate link in the Certification column.

### Environmental

The NI USB-845x 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](http://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](http://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](http://ni.com/environment/rohs_china)。(For information about China RoHS compliance, go to [ni.com/environment/rohs\\_china](http://ni.com/environment/rohs_china).)

---

# Technical Support and Professional Services

Log in to your National Instruments [ni.com](http://ni.com) User Profile to get personalized access to your services. Visit the following sections of [ni.com](http://ni.com) for technical support and professional services:

- **Support**—Technical support at [ni.com/support](http://ni.com/support) includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit [ni.com/support](http://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](http://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 eLearning training modules at [ni.com/elearning](http://ni.com/elearning). All customers automatically receive a one-year membership in the Standard Service Program (SSP) with the purchase of most software products and bundles including NI Developer Suite. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit [ni.com/ssp](http://ni.com/ssp) for more information.

For information about other technical support options in your area, visit [ni.com/services](http://ni.com/services), or contact your local office at [ni.com/contact](http://ni.com/contact).

- **Training and Certification**—Visit [ni.com/training](http://ni.com/training) for training and certification program information. You can also 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](http://ni.com/alliance).

You also can visit the Worldwide Offices section of [ni.com/niglobal](http://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	$10^{-9}$
$\mu$	micro	$10^{-6}$
m	milli	$10^{-3}$
k	kilo	$10^3$
M	mega	$10^6$
G	giga	$10^9$
T	tera	$10^{12}$

## A

**ACK Poll** The process of resending a slave address until it is acknowledged or a timeout occurs.

**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<sup>2</sup>C Inter-IC

## M

**Master** On the I<sup>2</sup>C 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<sup>2</sup>C 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

---

## Symbols

- +5 V power source
  - NI USB-8451, 3-11
  - NI USB-8452, 3-21

## A

- ACK polling, 1-4
- arbitration, 1-2

## B

- basic functions, 7-39, 13-34, 19-16
- basic VIs, 6-14, 12-13, 18-8
- block diagram (figure)
  - NI USB-8451, 3-2
  - NI USB-8452, 3-12
- bus interface specifications
  - NI USB-8451, A-4
  - NI USB-8452, A-12

## C

- C functions
  - basic functions, 7-39, 13-34, 19-16
  - configuration functions, 7-21, 10-18, 13-20, 16-16
  - data types, 7-1, 10-1, 13-1, 16-1, 19-1
  - general device functions, 7-8, 10-5, 13-8, 16-5, 19-4
  - ni845xClose, 7-8, 10-5, 13-8, 16-5, 19-4
  - ni845xCloseFindDeviceHandle, 7-9, 10-6, 13-9, 16-6, 19-5
  - ni845xDeviceLock, 7-10, 10-7, 13-10, 16-7, 19-6
  - ni845xDeviceUnlock, 7-11, 10-8, 13-11, 16-8, 19-7
  - ni845xDioReadLine, 19-16
  - ni845xDioReadPort, 19-18
  - ni845xDioSetDriverType, 19-20
  - ni845xDioSetPortLineDirectionMap, 19-19
  - ni845xDioWriteLine, 19-21
  - ni845xDioWritePort, 19-22
  - ni845xFindDevice, 7-12, 10-9, 13-12, 16-9, 19-8
  - ni845xFindDeviceNext, 7-14, 10-11, 13-14, 16-11, 19-10
  - ni845xI2cConfigurationClose, 7-21
  - ni845xI2cConfigurationGetAckPollTimeout, 7-22
  - ni845xI2cConfigurationGetAddress, 7-23
  - ni845xI2cConfigurationGetAddressSize, 7-24
  - ni845xI2cConfigurationGetClockRate, 7-25
  - ni845xI2cConfigurationGetHSClockRate, 7-26
  - ni845xI2cConfigurationGetHSEnable, 7-27
  - ni845xI2cConfigurationGetHSMasterCode, 7-28
  - ni845xI2cConfigurationGetPort, 7-29
  - ni845xI2cConfigurationOpen, 7-30
  - ni845xI2cConfigurationSetAckPollTimeout, 7-31
  - ni845xI2cConfigurationSetAddress, 7-32
  - ni845xI2cConfigurationSetAddressSize, 7-33
  - ni845xI2cConfigurationSetClockRate, 7-34
  - ni845xI2cConfigurationSetHSClockRate, 7-35
  - ni845xI2cConfigurationSetHSEnable, 7-36



- ni845xI2cConfigurationSetHSMasterCode, 7-37
- ni845xI2cConfigurationSetPort, 7-38
- ni845xI2cRead, 7-39
- ni845xI2cScriptAckPollTimeout, 7-45
- ni845xI2cScriptAddressRead, 7-46
- ni845xI2cScriptAddressWrite, 7-47
- ni845xI2cScriptClockRate, 7-48
- ni845xI2cScriptClose, 7-49
- ni845xI2cScriptDelay, 7-50
- ni845xI2cScriptDioConfigureLine, 7-51
- ni845xI2cScriptDioConfigurePort, 7-52
- ni845xI2cScriptDioReadLine, 7-53
- ni845xI2cScriptDioReadPort, 7-55
- ni845xI2cScriptDioWriteLine, 7-56
- ni845xI2cScriptDioWritePort, 7-58
- ni845xI2cScriptExtractReadData, 7-60
- ni845xI2cScriptExtractReadDataSize, 7-61
- ni845xI2cScriptHSClockRate, 7-64
- ni845xI2cScriptHSEnable, 7-62
- ni845xI2cScriptHSMasterCode, 7-63
- ni845xI2cScriptIssueStart, 7-65
- ni845xI2cScriptIssueStop, 7-66
- ni845xI2cScriptOpen, 7-67
- ni845xI2cScriptPullupEnable, 7-59
- ni845xI2cScriptRead, 7-68
- ni845xI2cScriptReset, 7-70
- ni845xI2cScriptRun, 7-71
- ni845xI2cScriptWrite, 7-73
- ni845xI2cScriptµsDelay, 7-72
- ni845xI2cSetPullupEnable, 7-17, 10-14
- ni845xI2cSlaveConfigurationClose, 10-18
- ni845xI2cSlaveConfigurationGetAddress, 10-19
- ni845xI2cSlaveConfigurationGetAutoRemovalTimeout, 10-20
- ni845xI2cSlaveConfigurationGetCommandBufferSize, 10-21
- ni845xI2cSlaveConfigurationGetDataBufferSize, 10-22
- ni845xI2cSlaveConfigurationOpen, 10-23
- ni845xI2cSlaveConfigurationSetAddress, 10-24
- ni845xI2cSlaveConfigurationSetAutoRemovalTimeout, 10-25
- ni845xI2cSlaveConfigurationSetCommandBufferSize, 10-26
- ni845xI2cSlaveConfigurationSetDataBufferSize, 10-27
- ni845xI2cSlaveGetWriteInfo, 10-28
- ni845xI2cSlaveRead, 10-29
- ni845xI2cSlaveStart, 10-31
- ni845xI2cSlaveStop, 10-32
- ni845xI2cSlaveWaitForEvent, 10-33
- ni845xI2cSlaveWrite, 10-36
- ni845xI2cWrite, 7-41
- ni845xI2cWriteRead, 7-43
- ni845xOpen, 7-15, 10-12, 13-15, 16-12, 19-11
- ni845xSetIoVoltageLevel, 7-16, 10-13, 13-16, 19-12
- ni845xSetTimeout, 7-20, 10-15, 13-17, 16-13, 19-13
- ni845xSpiConfigurationClose, 13-20
- ni845xSpiConfigurationGetChipSelect, 13-21
- ni845xSpiConfigurationGetClockPhase, 13-22
- ni845xSpiConfigurationGetClockPolarity, 13-23
- ni845xSpiConfigurationGetClockRate, 13-24
- ni845xSpiConfigurationGetNumBitsPerSample, 13-25
- ni845xSpiConfigurationGetPort, 13-26
- ni845xSpiConfigurationOpen, 13-27
- ni845xSpiConfigurationSetChipSelect, 13-28

- ni845xSpiConfigurationSetClockPhase, 13-29
- ni845xSpiConfigurationSetClockPolarity, 13-30
- ni845xSpiConfigurationSetClockRate, 13-31
- ni845xSpiConfigurationSetNumBitsPerSample, 13-32
- ni845xSpiConfigurationSetPort, 13-33
- ni845xSpiScriptClockPolarityPhase, 13-36
- ni845xSpiScriptClockRate, 13-38
- ni845xSpiScriptClose, 13-39
- ni845xSpiScriptCSHigh, 13-40
- ni845xSpiScriptCSLow, 13-41
- ni845xSpiScriptDelay, 13-42
- ni845xSpiScriptDioConfigureLine, 13-43
- ni845xSpiScriptDioConfigurePort, 13-44
- ni845xSpiScriptDioReadLine, 13-45
- ni845xSpiScriptDioReadPort, 13-47
- ni845xSpiScriptDioWriteLine, 13-48
- ni845xSpiScriptDioWritePort, 13-50
- ni845xSpiScriptDisableSPI, 13-51
- ni845xSpiScriptEnableSPI, 13-52
- ni845xSpiScriptExtractReadData, 13-53
- ni845xSpiScriptExtractReadDataSize, 13-54
- ni845xSpiScriptNumBitsPerSample, 13-55
- ni845xSpiScriptOpen, 13-56
- ni845xSpiScriptReset, 13-57
- ni845xSpiScriptRun, 13-58
- ni845xSpiScriptUsDelay, 13-59
- ni845xSpiScriptWriteRead, 13-60
- ni845xSpiStreamConfigurationClose, 16-16
- ni845xSpiStreamConfigurationGetClockPhase, 16-21
- ni845xSpiStreamConfigurationGetClockPolarity, 16-23
- ni845xSpiStreamConfigurationGetNumBits, 16-18
- ni845xSpiStreamConfigurationGetNumSamples, 16-19
- ni845xSpiStreamConfigurationGetPacketSize, 16-20
- ni845xSpiStreamConfigurationOpen, 16-17
- ni845xSpiStreamConfigurationSetClockPhase, 16-31
- ni845xSpiStreamConfigurationSetClockPolarity, 16-33
- ni845xSpiStreamConfigurationSetNumBits, 16-28
- ni845xSpiStreamConfigurationSetNumSamples, 16-29
- ni845xSpiStreamConfigurationSetPacketSize, 16-30
- ni845xSpiStreamConfigurationWave1GetPinConfig, 16-22
- ni845xSpiStreamConfigurationWave1GetTimingParam, 16-24
- ni845xSpiStreamConfigurationWave1SetMosiData, 16-26
- ni845xSpiStreamConfigurationWave1SetPinConfig, 16-32
- ni845xSpiStreamConfigurationWave1SetTimingParam, 16-34
- ni845xSpiStreamRead, 16-36
- ni845xSpiStreamStart, 16-38
- ni845xSpiStreamStop, 16-39
- ni845xSpiWriteRead, 13-34
- ni845xStatusToString, 7-18, 10-16, 13-18, 16-14, 19-14
- scripting functions, 7-45, 13-36
- section headings, 7-1, 10-1, 13-1, 16-1, 19-1
- slave functions, 10-28
- streaming functions, 16-36
- CE compliance specifications, A-15
- Chip Select pin, 14-4
- clock and polarity, 1-7

- clock stretching, 1-4
- configuration functions, 7-21, 10-18, 13-20, 16-16
- configuration VIs, 6-8, 9-8, 12-8, 15-8
- CONV pin, 14-4
- conventions used in the manual, *xxi*
- current levels, 1-6

## D

- diagnostic tools (NI resources), B-1
- digital I/O
  - NI USB-8451, 3-7
  - NI USB-8452, 3-19
  - specifications
    - NI USB-8451, A-1
    - NI USB-8452, A-6
- digital terminal assignments (figure)
  - NI USB-8451, 3-4
- dimensions
  - front and rear view (figure)
    - NI USB-8451 OEM, A-5
    - NI USB-8452 OEM, A-14
  - top view (figure)
    - NI USB-8451 OEM, A-5
    - NI USB-8452 OEM, A-13
- DIO line read, 17-2
- DIO line write, 17-2
- DIO port configure, 17-2
- DIO port read, 17-2
- DIO port write, 17-2
- documentation
  - conventions used in manual, *xxi*
  - NI resources, B-1
- DRDY pin, 14-4
- drivers (NI resources), B-1

## E

- electromagnetic compatibility
  - specifications, A-14
- environmental management specifications, A-16
- environmental specifications, A-16
- error handling, 1-8
- examples (NI resources), B-1
- extended (10-bit) addressing, 1-4
- external user-provided resistor, connection
  - example (figure)
    - NI USB-8451, 3-7
- extract read data
  - I<sup>2</sup>C API, 5-6
  - SPI API, 11-6

## G

- general device functions, 7-8, 10-5, 13-8, 16-5, 19-4
- general device VIs, 6-2, 9-2, 12-2, 15-2, 18-2

## H

- hardware
  - installation, 2-1
  - overview, 3-1
    - NI USB-8451, 3-1
    - NI USB-8452, 3-11
  - setup
    - NI USB-8451, 3-2
    - NI USB-8451 OEM, 3-3
    - NI USB-8452, 3-13
  - specifications, A-1
- hazardous locations, A-14
- help, technical support, B-1

**I**

- I/O connector and cable
  - NI USB-8451, 3-4
  - NI USB-8451 OEM, 3-5
- I/O protection
  - NI USB-8451, 3-10
  - NI USB-8452, 3-21
- I<sup>2</sup>C bus, 1-1, 1-2
  - ACK polling, 1-4
  - arbitration, 1-2
  - clock stretching, 1-4
  - extended (10-bit) addressing, 1-4
  - High Speed master code, 1-5
  - terminology, 1-1
  - transfers (figure), 1-3
- I<sup>2</sup>C configure, 5-2
- I<sup>2</sup>C interface
  - NI USB-8451, 3-10
  - NI USB-8452, 3-18
  - specifications
    - NI USB-8451, A-3
    - NI USB-8452, A-10
  - to two peripherals (figure)
    - NI USB-8451, 3-10
    - NI USB-8452, 3-19
- I<sup>2</sup>C read, 5-2
- I<sup>2</sup>C slave configure, 8-2
- I<sup>2</sup>C slave get write information, 8-3
- I<sup>2</sup>C slave read, 8-3
- I<sup>2</sup>C slave start, 8-2
- I<sup>2</sup>C slave stop, 8-3
- I<sup>2</sup>C slave wait for event, 8-3
- I<sup>2</sup>C slave write, 8-3
- I<sup>2</sup>C vs. SMBus, 1-5
  - current levels, 1-6
  - logic levels, 1-5
  - timeout and clock rates, 1-5
- I<sup>2</sup>C write, 5-2
- I<sup>2</sup>C write read, 5-2

## installation

- hardware, 2-1
- software, 2-1
- instrument drivers (NI resources), B-1
- introduction, 1-1

**K**

- KnowledgeBase, B-1

**L**

## LabVIEW VIs

- basic VIs, 6-14, 12-13, 18-8
- configuration VIs, 6-8, 9-8, 12-8, 15-8
- general device VIs, 6-2, 9-2, 12-2, 15-2, 18-2
- NI-845x Close Reference.vi, 6-2, 9-2, 12-2, 15-2, 18-2
- NI-845x Device Property Node, 6-4, 9-4, 12-4, 15-4, 18-4
- NI-845x Device Reference, 6-7, 9-7, 12-7, 15-7, 18-7
- NI-845x DIO Read Line.vi, 18-8
- NI-845x DIO Read Port.vi, 18-10
- NI-845x DIO Write Line.vi, 18-12
- NI-845x DIO Write Port.vi, 18-14
- NI-845x I2C Configuration Property Node, 6-8
- NI-845x I2C Create Configuration Reference.vi, 6-12
- NI-845x I2C Create Script Reference.vi, 6-20
- NI-845x I2C Extract Script Read Data.vi, 6-22
- NI-845x I2C Read.vi, 6-14
- NI-845x I2C Run Script.vi, 6-24
- NI-845x I2C Script ACK Poll Timeout.vi, 6-26
- NI-845x I2C Script Address+Read.vi, 6-28

- NI-845x I2C Script Address+Write.vi,  
6-30
- NI-845x I2C Script Clock Rate.vi, 6-32
- NI-845x I2C Script Delay  
(Microsecond).vi, 6-34
- NI-845x I2C Script Delay  
(Millisecond).vi, 6-36
- NI-845x I2C Script DIO Configure  
Line.vi, 6-38
- NI-845x I2C Script DIO Configure  
Port.vi, 6-40
- NI-845x I2C Script DIO Read Line.vi,  
6-42
- NI-845x I2C Script DIO Read Port.vi,  
6-44
- NI-845x I2C Script DIO Write Line.vi,  
6-46
- NI-845x I2C Script DIO Write Port.vi,  
6-48
- NI-845x I2C Script HS Clock Rate.vi,  
6-56
- NI-845x I2C Script HS Enable.vi, 6-52
- NI-845x I2C Script HS Master Code.vi,  
6-54
- NI-845x I2C Script Issue Start.vi, 6-58
- NI-845x I2C Script Issue Stop.vi, 6-60
- NI-845x I2C Script Pullup Enable.vi, 6-50
- NI-845x I2C Script Read.vi, 6-62
- NI-845x I2C Script Write.vi, 6-64
- NI-845x I2C Slave Configuration  
Property Node, 9-8
- NI-845x I2C Slave Create Configuration  
Reference.vi, 9-11
- NI-845x I2C Slave Get Write Info.vi, 9-13
- NI-845x I2C Slave Read.vi, 9-15
- NI-845x I2C Slave Start.vi, 9-17
- NI-845x I2C Slave Stop.vi, 9-19
- NI-845x I2C Slave Wait for Event.vi, 9-21
- NI-845x I2C Slave Write.vi, 9-24
- NI-845x I2C Write Read.vi, 6-18
- NI-845x I2C Write.vi, 6-16
- NI-845x SPI Configuration Property  
Node, 12-8
- NI-845x SPI Create Configuration  
Reference.vi, 12-11
- NI-845x SPI Create Script Reference.vi,  
12-16
- NI-845x SPI Extract Script Read Data.vi,  
12-18
- NI-845x SPI Run Script.vi, 12-20
- NI-845x SPI Script Clock Polarity  
Phase.vi, 12-22
- NI-845x SPI Script Clock Rate.vi, 12-24
- NI-845x SPI Script CS High.vi, 12-26
- NI-845x SPI Script CS Low.vi, 12-28
- NI-845x SPI Script Delay  
(Microsecond).vi, 12-30
- NI-845x SPI Script Delay  
(Millisecond).vi, 12-32
- NI-845x SPI Script DIO Configure  
Line.vi, 12-34
- NI-845x SPI Script DIO Configure  
Port.vi, 12-36
- NI-845x SPI Script DIO Read Line.vi,  
12-38
- NI-845x SPI Script DIO Read Port.vi,  
12-40
- NI-845x SPI Script DIO Write Line.vi,  
12-42
- NI-845x SPI Script DIO Write Port.vi,  
12-44
- NI-845x SPI Script Disable SPI.vi, 12-46
- NI-845x SPI Script Enable SPI.vi, 12-48
- NI-845x SPI Script Num Bits Per  
Sample.vi, 12-50
- NI-845x SPI Script Write Read.vi, 12-52
- NI-845x SPI Stream Configuration  
Property Node, 15-8
- NI-845x SPI Stream Create Configuration  
Reference.vi, 15-16
- NI-845x SPI Stream Read.vi, 15-18,  
15-20, 15-22
- NI-845x SPI Stream Start.vi, 15-20

- NI-845x SPI Stream Stop.vi, 15-22
- NI-845x SPI Write Read.vi, 12-13
- scripting VIs, 6-20, 12-16
- streaming VIs, 15-18

#### LED indicators

- NI USB-8452 (figure), 3-20
- NI USB-8452 OEM, 3-20

#### load connection example (figure)

- NI USB-8451, 3-8

#### logic levels, 1-5

## N

- National Instruments support and services, B-1

- NI support and services, B-1

#### NI USB-8451, 3-1

- +5 V power source, 3-11
- block diagram (figure), 3-2
- digital I/O, 3-7
- digital terminal assignments (figure), 3-4
- external user-provided resistor,
  - connection example (figure), 3-7
- hardware
  - overview, 3-1
  - setup, 3-2
- I/O connector and cable, 3-4
- I/O protection, 3-10
- I<sup>2</sup>C interface, 3-10
  - to two peripherals (figure), 3-10
- load connection example (figure), 3-8
- power-on states, 3-11
- signal descriptions (table), 3-6
- signal label application diagram (figure), 3-3
- software installation, 3-2
- specifications, A-1
- SPI interface, 3-9
  - to three peripherals (figure), 3-9

#### NI USB-8451 OEM

- hardware
  - setup, 3-3
- I/O connector and cable, 3-5
- pin assignments (table), 3-5

#### NI USB-8452, 3-11

- +5 V power source, 3-21
- block diagram (figure), 3-12
- digital I/O, 3-19
- hardware
  - overview, 3-11
  - setup, 3-13
- I/O protection, 3-21
- I<sup>2</sup>C interface, 3-18
  - to two peripherals (figure), 3-19
- LED indicators (figure), 3-20
- pin assignments (figure), 3-14
- power-on states, 3-21
- signal descriptions (table), 3-14
- software installation, 3-13
- specifications, A-6
- SPI interface, 3-16
  - standard mode, 3-18
  - stream mode, 3-18
  - to three peripherals (figure), 3-17
- V<sub>ref</sub> I/O reference voltage, 3-22
- with enclosure (figure), 3-12

#### NI USB-8452 OEM

- LED indicators (figure), 3-20

#### NI-845x API, 4-1

- NI-845x Close Reference.vi, 6-2, 9-2, 12-2, 15-2, 18-2
- NI-845x Device Property Node, 6-4, 9-4, 12-4, 15-4, 18-4
- NI-845x Device Reference, 6-7, 9-7, 12-7, 15-7, 18-7
- NI-845x DIO API, 17-1
  - basic programming model, 17-1
  - DIO line read, 17-2
  - DIO line write, 17-2
  - DIO port configure, 17-2

- DIO port read, 17-2
- DIO port write, 17-2
- C functions, 19-1
- LabVIEW VIs, 18-1
- list of C functions, 19-2
- NI-845x DIO Read Line.vi, 18-8
- NI-845x DIO Read Port.vi, 18-10
- NI-845x DIO Write Line.vi, 18-12
- NI-845x DIO Write Port.vi, 18-14
- NI-845x I<sup>2</sup>C API, 5-1
  - advanced programming model
    - extract read data, 5-6
    - run script, 5-6
    - script: issue start condition, 5-5
    - script: issue stop condition, 5-6
    - script: pullup enable, 5-4
    - script: read, 5-5
    - script: send address + read, 5-5
    - script: send address + write, 5-5
    - script: send High Speed master code, 5-5
    - script: set I<sup>2</sup>C ACK poll timeout, 5-4
    - script: set I<sup>2</sup>C clock rate, 5-4
    - script: set I<sup>2</sup>C High Speed clock rate, 5-4
    - script: set I<sup>2</sup>C High Speed enable, 5-4
    - script: write, 5-6
  - basic programming model, 5-1
    - I<sup>2</sup>C configure, 5-2
    - I<sup>2</sup>C read, 5-2
    - I<sup>2</sup>C write, 5-2
    - I<sup>2</sup>C write read, 5-2
  - C functions, 7-1
  - LabVIEW VIs, 6-1, 9-1
  - list of C functions, 7-2, 10-2
  - scripting programming model, 5-2
    - example (figure), 5-3
- NI-845x I2C Configuration Property Node, 6-8
- NI-845x I2C Create Configuration Reference.vi, 6-12
- NI-845x I2C Create Script Reference.vi, 6-20
- NI-845x I2C Extract Script Read Data.vi, 6-22
- NI-845x I2C Read.vi, 6-14
- NI-845x I2C Run Script.vi, 6-24
- NI-845x I2C Script ACK Poll Timeout.vi, 6-26
- NI-845x I2C Script Address+Read.vi, 6-28
- NI-845x I2C Script Address+Write.vi, 6-30
- NI-845x I2C Script Clock Rate.vi, 6-32
- NI-845x I2C Script Delay (Microsecond).vi, 6-34
- NI-845x I2C Script Delay (Millisecond).vi, 6-36
- NI-845x I2C Script DIO Configure Line.vi, 6-38
- NI-845x I2C Script DIO Configure Port.vi, 6-40
- NI-845x I2C Script DIO Read Line.vi, 6-42
- NI-845x I2C Script DIO Read Port.vi, 6-44
- NI-845x I2C Script DIO Write Line.vi, 6-46
- NI-845x I2C Script DIO Write Port.vi, 6-48
- NI-845x I2C Script HS Clock Rate.vi, 6-56
- NI-845x I2C Script HS Enable.vi, 6-52
- NI-845x I2C Script HS Master Code.vi, 6-54
- NI-845x I2C Script Issue Start.vi, 6-58
- NI-845x I2C Script Issue Stop.vi, 6-60
- NI-845x I2C Script Pullup Enable.vi, 6-50
- NI-845x I2C Script Read.vi, 6-62
- NI-845x I2C Script Write.vi, 6-64
- NI-845x I<sup>2</sup>C Slave API, 8-1
  - C functions, 10-1
  - programming model, 8-1
    - I<sup>2</sup>C slave configure, 8-2
    - I<sup>2</sup>C slave get write information, 8-3
    - I<sup>2</sup>C slave read, 8-3
    - I<sup>2</sup>C slave start, 8-2
    - I<sup>2</sup>C slave stop, 8-3
    - I<sup>2</sup>C slave wait for event, 8-3
    - I<sup>2</sup>C slave write, 8-3
- NI-845x I2C Slave Configuration Property Node, 9-8

- NI-845x I2C Slave Create Configuration Reference.vi, 9-11
- NI-845x I2C Slave Get Write Info.vi, 9-13
- NI-845x I2C Slave Read.vi, 9-15
- NI-845x I2C Slave Start.vi, 9-17
- NI-845x I2C Slave Stop.vi, 9-19
- NI-845x I2C Slave Wait for Event.vi, 9-21
- NI-845x I2C Slave Write.vi, 9-24
- NI-845x I2C Write Read.vi, 6-18
- NI-845x I2C Write.vi, 6-16
- NI-845x SPI API, 11-1
  - advanced programming model
    - extract read data, 11-6
    - run script, 11-6
    - script: chip select high, 11-6
    - script: chip select low, 11-5
    - script: configure phase, polarity, clock rate, number of bits, 11-5
    - script: disable SPI, 11-6
    - script: enable SPI, 11-5
    - script: write read, 11-5
    - scripting functions programming example (figure), 11-4
  - basic programming model, 11-1
    - SPI configure, 11-2
    - SPI timing characteristics, 11-2
    - SPI write read, 11-2
  - C functions, 13-1
  - LabVIEW VIs, 12-1
  - list of C functions, 13-2
  - scripting programming model, 11-3
- NI-845x SPI Configuration Property Node, 12-8
- NI-845x SPI Create Configuration Reference.vi, 12-11
- NI-845x SPI Create Script Reference.vi, 12-16
- NI-845x SPI Extract Script Read Data.vi, 12-18
- NI-845x SPI Run Script.vi, 12-20
- NI-845x SPI Script Clock Polarity Phase.vi, 12-22
- NI-845x SPI Script Clock Rate.vi, 12-24
- NI-845x SPI Script CS High.vi, 12-26
- NI-845x SPI Script CS Low.vi, 12-28
- NI-845x SPI Script Delay (Microsecond).vi, 12-30
- NI-845x SPI Script Delay (Millisecond).vi, 12-32
- NI-845x SPI Script DIO Configure Line.vi, 12-34
- NI-845x SPI Script DIO Configure Port.vi, 12-36
- NI-845x SPI Script DIO Read Line.vi, 12-38
- NI-845x SPI Script DIO Read Port.vi, 12-40
- NI-845x SPI Script DIO Write Line.vi, 12-42
- NI-845x SPI Script DIO Write Port.vi, 12-44
- NI-845x SPI Script Disable SPI.vi, 12-46
- NI-845x SPI Script Enable SPI.vi, 12-48
- NI-845x SPI Script Num Bits Per Sample.vi, 12-50
- NI-845x SPI Script Write Read.vi, 12-52
- NI-845x SPI Stream API
  - C functions, 16-1
  - Chip Select pin, 14-4
  - CONV pin, 14-4
  - DRDY pin, 14-4
  - extra SPI pin descriptions, 14-4
  - LabVIEW VIs, 15-1
  - list of C functions, 16-2
  - programming model, 14-1
  - SPI stream configure, 14-2
  - SPI stream read, 14-2
  - SPI stream start, 14-2
  - SPI stream stop, 14-2
  - using, 14-1
  - waveform 1, 14-3
    - timing diagram (figure), 14-3
    - timing parameters (table), 14-4
- NI-845x SPI Stream Configuration Property Node, 15-8
- NI-845x SPI Stream Create Configuration Reference.vi, 15-16



- NI-845x SPI Stream Read.vi, 15-18, 15-20, 15-22
- NI-845x SPI Stream Start.vi, 15-20
- NI-845x SPI Stream Stop.vi, 15-22
- NI-845x SPI Write Read.vi, 12-13
- ni845xClose, 7-8, 10-5, 13-8, 16-5, 19-4
- ni845xCloseFindDeviceHandle, 7-9, 10-6, 13-9, 16-6, 19-5
- ni845xDeviceLock, 7-10, 10-7, 13-10, 16-7, 19-6
- ni845xDeviceUnlock, 7-11, 10-8, 13-11, 16-8, 19-7
- ni845xDioReadLine, 19-16
- ni845xDioReadPort, 19-18
- ni845xDioSetDriverType, 19-20
- ni845xDioSetPortLineDirectionMap, 19-19
- ni845xDioWriteLine, 19-21
- ni845xDioWritePort, 19-22
- ni845xFindDevice, 7-12, 10-9, 13-12, 16-9, 19-8
- ni845xFindDeviceNext, 7-14, 10-11, 13-14, 16-11, 19-10
- ni845xI2cConfigurationClose, 7-21
- ni845xI2cConfigurationGetAckPollTimeout, 7-22
- ni845xI2cConfigurationGetAddress, 7-23
- ni845xI2cConfigurationGetAddressSize, 7-24
- ni845xI2cConfigurationGetClockRate, 7-25
- ni845xI2cConfigurationGetHSClockRate, 7-26
- ni845xI2cConfigurationGetHSEnable, 7-27
- ni845xI2cConfigurationGetHSMasterCode, 7-28
- ni845xI2cConfigurationGetPort, 7-29
- ni845xI2cConfigurationOpen, 7-30
- ni845xI2cConfigurationSetAckPollTimeout, 7-31
- ni845xI2cConfigurationSetAddress, 7-32
- ni845xI2cConfigurationSetAddressSize, 7-33
- ni845xI2cConfigurationSetClockRate, 7-34
- ni845xI2cConfigurationSetHSClockRate, 7-35
- ni845xI2cConfigurationSetHSEnable, 7-36
- ni845xI2cConfigurationSetHSMasterCode, 7-37
- ni845xI2cConfigurationSetPort, 7-38
- ni845xI2cRead, 7-39
- ni845xI2cScriptAckPollTimeout, 7-45
- ni845xI2cScriptAddressRead, 7-46
- ni845xI2cScriptAddressWrite, 7-47
- ni845xI2cScriptClockRate, 7-48
- ni845xI2cScriptClose, 7-49
- ni845xI2cScriptDelay, 7-50
- ni845xI2cScriptDioConfigureLine, 7-51
- ni845xI2cScriptDioConfigurePort, 7-52
- ni845xI2cScriptDioReadLine, 7-53
- ni845xI2cScriptDioReadPort, 7-55
- ni845xI2cScriptDioWriteLine, 7-56
- ni845xI2cScriptDioWritePort, 7-58
- ni845xI2cScriptExtractReadData, 7-60
- ni845xI2cScriptExtractReadDataSize, 7-61
- ni845xI2cScriptHSClockRate, 7-64
- ni845xI2cScriptHSEnable, 7-62
- ni845xI2cScriptHSMasterCode, 7-63
- ni845xI2cScriptIssueStart, 7-65
- ni845xI2cScriptIssueStop, 7-66
- ni845xI2cScriptOpen, 7-67
- ni845xI2cScriptPullupEnable, 7-59
- ni845xI2cScriptRead, 7-68
- ni845xI2cScriptReset, 7-70
- ni845xI2cScriptRun, 7-71
- ni845xI2cScriptUsDelay, 7-72
- ni845xI2cScriptWrite, 7-73
- ni845xI2cSetPullupEnable, 7-17, 10-14
- ni845xI2cSlaveConfigurationClose, 10-18
- ni845xI2cSlaveConfigurationGetAddress, 10-19
- ni845xI2cSlaveConfigurationGetAutoRemovalTimeout, 10-20
- ni845xI2cSlaveConfigurationGetCommandBufferSize, 10-21

- ni845xI2cSlaveConfigurationGetDataBufferSize, 10-22
- ni845xI2cSlaveConfigurationOpen, 10-23
- ni845xI2cSlaveConfigurationSetAddress, 10-24
- ni845xI2cSlaveConfigurationSetAutoRemovalTimeout, 10-25
- ni845xI2cSlaveConfigurationSetCommandBufferSize, 10-26
- ni845xI2cSlaveConfigurationSetDataBufferSize, 10-27
- ni845xI2cSlaveGetWriteInfo, 10-28
- ni845xI2cSlaveRead, 10-29
- ni845xI2cSlaveStart, 10-31
- ni845xI2cSlaveStop, 10-32
- ni845xI2cSlaveWaitForEvent, 10-33
- ni845xI2cSlaveWrite, 10-36
- ni845xI2cWrite, 7-41
- ni845xI2cWriteRead, 7-43
- ni845xOpen, 7-15, 10-12, 13-15, 16-12, 19-11
- ni845xSetIoVoltageLevel, 7-16, 10-13, 13-16, 19-12
- ni845xSetTimeout, 7-20, 10-15, 13-17, 16-13, 19-13
- ni845xSpiConfigurationClose, 13-20
- ni845xSpiConfigurationGetChipSelect, 13-21
- ni845xSpiConfigurationGetClockPhase, 13-22
- ni845xSpiConfigurationGetClockPolarity, 13-23
- ni845xSpiConfigurationGetClockRate, 13-24
- ni845xSpiConfigurationGetNumBitsPerSample, 13-25
- ni845xSpiConfigurationGetPort, 13-26
- ni845xSpiConfigurationOpen, 13-27
- ni845xSpiConfigurationSetChipSelect, 13-28
- ni845xSpiConfigurationSetClockPhase, 13-29
- ni845xSpiConfigurationSetClockPolarity, 13-30
- ni845xSpiConfigurationSetClockRate, 13-31
- ni845xSpiConfigurationSetNumBitsPerSample, 13-32
- ni845xSpiConfigurationSetPort, 13-33
- ni845xSpiScriptClockPolarityPhase, 13-36
- ni845xSpiScriptClockRate, 13-38
- ni845xSpiScriptClose, 13-39
- ni845xSpiScriptCSHigh, 13-40
- ni845xSpiScriptCSLow, 13-41
- ni845xSpiScriptDelay, 13-42
- ni845xSpiScriptDioConfigureLine, 13-43
- ni845xSpiScriptDioConfigurePort, 13-44
- ni845xSpiScriptDioReadLine, 13-45
- ni845xSpiScriptDioReadPort, 13-47
- ni845xSpiScriptDioWriteLine, 13-48
- ni845xSpiScriptDioWritePort, 13-50
- ni845xSpiScriptDisableSPI, 13-51
- ni845xSpiScriptEnableSPI, 13-52
- ni845xSpiScriptExtractReadData, 13-53
- ni845xSpiScriptExtractReadDataSize, 13-54
- ni845xSpiScriptNumBitsPerSample, 13-55
- ni845xSpiScriptOpen, 13-56
- ni845xSpiScriptReset, 13-57
- ni845xSpiScriptRun, 13-58
- ni845xSpiScriptUsDelay, 13-59
- ni845xSpiScriptWriteRead, 13-60
- ni845xSpiStreamConfigurationClose, 16-16
- ni845xSpiStreamConfigurationGetClockPhase, 16-21
- ni845xSpiStreamConfigurationGetClockPolarity, 16-23
- ni845xSpiStreamConfigurationGetNumBits, 16-18
- ni845xSpiStreamConfigurationGetNumSamples, 16-19
- ni845xSpiStreamConfigurationGetPacketSize, 16-20
- ni845xSpiStreamConfigurationOpen, 16-17
- ni845xSpiStreamConfigurationSetClockPhase, 16-31
- ni845xSpiStreamConfigurationSetClockPolarity, 16-33

- ni845xSpiStreamConfigurationSetNumBits, 16-28
- ni845xSpiStreamConfigurationSetNumSamples, 16-29
- ni845xSpiStreamConfigurationSetPacketSize, 16-30
- ni845xSpiStreamConfigurationWave1GetPinConfig, 16-22
- ni845xSpiStreamConfigurationWave1GetTimingParam, 16-24
- ni845xSpiStreamConfigurationWave1SetMosiData, 16-26
- ni845xSpiStreamConfigurationWave1SetPinConfig, 16-32
- ni845xSpiStreamConfigurationWave1SetTimingParam, 16-34
- ni845xSpiStreamRead, 16-36
- ni845xSpiStreamStart, 16-38
- ni845xSpiStreamStop, 16-39
- ni845xSpiWriteRead, 13-34
- ni845xStatusToString, 7-18, 10-16, 13-18, 16-14, 19-14

## O

- online product certification specifications, A-16
- output voltage source specifications
  - NI USB-8451, A-4
  - NI USB-8452, A-12
- overvoltage protection specifications
  - NI USB-8451, A-6

## P

- physical characteristic specifications
  - NI USB-8451, A-4
  - NI USB-8451 OEM, A-5
  - NI USB-8452, A-12
  - NI USB-8452 OEM, A-13

- pin assignments
  - NI USB-8451 OEM, 3-5
  - NI USB-8452, 3-14
- power requirements specifications
  - NI USB-8451, A-4
  - NI USB-8452, A-12
- power-on states
  - NI USB-8451, 3-11
  - NI USB-8452, 3-21
- programming examples (NI resources), B-1

## R

- run script
  - I<sup>2</sup>C API, 5-6
  - SPI API, 11-6

## S

- safety specifications, A-14
- scripting functions, 7-45, 13-36
- scripting VIs, 6-20, 12-16
- scripts
  - chip select high, 11-6
  - chip select low, 11-5
  - configure phase, polarity, clock rate, number of bits, 11-5
  - disable SPI, 11-6
  - enable SPI, 11-5
  - issue start condition, 5-5
  - issue stop condition, 5-6
  - pullup enable, 5-4
  - read, 5-5
  - send address + read, 5-5
  - send address + write, 5-5
  - send High Speed master code, 5-5
  - set I<sup>2</sup>C ACK poll timeout, 5-4
  - set I<sup>2</sup>C clock rate, 5-4
  - set I<sup>2</sup>C High Speed clock rate, 5-4
  - set I<sup>2</sup>C High Speed Enable, 5-4

- write, 5-6
- write read, 11-5
- signal descriptions (table)
  - NI USB-8451, 3-6
  - NI USB-8452, 3-14
- signal label application diagram (figure), 3-3
- slave functions, 10-28
- SMBus
  - current levels, 1-6
  - logic levels, 1-5
  - timeout and clock rates, 1-5
- software
  - installation, 2-1
    - NI USB-8451, 3-2
    - NI USB-8452, 3-13
  - NI resources, B-1
- specifications, A-1
  - CE compliance, A-15
  - electromagnetic compatibility, A-14
  - environmental, A-16
  - environmental management, A-16
  - NI USB-8451, A-1
    - bus interface, A-4
    - digital I/O, A-1
    - I<sup>2</sup>C interface, A-3
    - output voltage sources, A-4
    - overvoltage protection, A-6
    - physical characteristics, A-4
    - power requirements, A-4
    - SPI interface, A-2
  - NI USB-8451 OEM
    - front and rear view dimensions (figure), A-5
    - physical characteristics, A-5
    - top view dimensions (figure), A-5
  - NI USB-8452, A-6
    - bus interface, A-12
    - digital I/O, A-6
    - I<sup>2</sup>C interface, A-10
    - output voltage sources, A-12
    - physical characteristics, A-12
    - power requirements, A-12
    - SPI interface, A-7
  - NI USB-8452 OEM
    - front and rear view dimensions (figure), A-14
    - physical characteristics, A-13
    - top view dimensions (figure), A-13
  - online product certification, A-16
  - safety, A-14
    - hazardous locations, A-14
- SPI bus, 1-6
  - clock and polarity, 1-7
  - error handling, 1-8
  - overview, 1-7
  - terminology, 1-6
- SPI configure, 11-2
- SPI interface
  - NI USB-8451, 3-9
  - NI USB-8452, 3-16
  - specifications
    - NI USB-8451, A-2
    - NI USB-8452, A-7
  - standard mode, NI USB-8452, 3-18
  - stream mode, NI USB-8452, 3-18
  - to three peripherals (figure)
    - NI USB-8451, 3-9
    - NI USB-8452, 3-17
- SPI pin descriptions, 14-4
- SPI Stream API, 14-1
  - extra pin descriptions, 14-4
  - programming model, 14-1
  - waveform 1, 14-3
- SPI stream configure, 14-2
- SPI stream read, 14-2
- SPI stream start, 14-2
- SPI stream stop, 14-2
- SPI timing characteristics, 11-2
- SPI write read, 11-2
- streaming functions, 16-36
- streaming VIs, 15-18

support, technical, B-1  
system configuration API, 2-3

## **T**

technical support, B-1  
terminology

- I<sup>2</sup>C bus, 1-1
- SPI bus, 1-6

timeout and clock rates, 1-5  
training and certification (NI resources), B-1  
troubleshooting (NI resources), B-1

## **V**

Vref I/O reference voltage  
NI USB-8452, 3-22

## **W**

waveform 1

- timing diagram (figure), 14-3
- timing parameters (table), 14-4

waveform1, 14-3  
Web resources, B-1