CAN

NI-CAN™ Programmer Reference Manual

November 2001 Edition
Part Number 370289C-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838,
China (ShenZhen) 0755 3904939, Czech Republic 02 2423 5774, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186,
India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Malaysia 603 9596711, Mexico 001 800 010 0793, Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011, Russia 095 2387139,
Singapore 2265886, Slovenia 386 3 425 4200, South Africa 11 805 8197, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

Patents

The product described in this manual may be protected by one or more U.S. patents: U.S. Patent No. 5,938,754.

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

FCC/Canada Radio Frequency Interference Compliance*

Determining FCC Class

The Federal Communications Commission (FCC) has rules to protect wireless communications from interference. The FCC places digital electronics into two classes. These classes are known as Class A (for use in industrial-commercial locations only) or Class B (for use in residential or commercial locations). Depending on where it is operated, this product could be subject to restrictions in the FCC rules. (In Canada, the Department of Communications (DOC), of Industry Canada, regulates wireless interference in much the same way.)

Digital electronics emit weak signals during normal operation that can affect radio, television, or other wireless products. By examining the product you purchased, you can determine the FCC Class and therefore which of the two FCC/DOC Warnings apply in the following sections. (Some products may not be labeled at all for FCC; if so, the reader should then assume these are Class A devices.)

FCC Class A products only display a simple warning statement of one paragraph in length regarding interference and undesired operation. Most of our products are FCC Class A. The FCC rules have restrictions regarding the locations where FCC Class A products can be operated.

FCC Class B products display either a FCC ID code, starting with the letters **EXN**, or the FCC Class B compliance mark that appears as shown here on the right.

Consult the FCC web site `http://www.fcc.gov` for more information.

FCC/DOC Warnings

This equipment generates and uses radio frequency energy and, if not installed and used in strict accordance with the instructions in this manual and the CE Mark Declaration of Conformity**, may cause interference to radio and television reception. Classification requirements are the same for the Federal Communications Commission (FCC) and the Canadian Department of Communications (DOC).

Changes or modifications not expressly approved by National Instruments could void the user's authority to operate the equipment under the FCC Rules.

Class A

Federal Communications Commission

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Canadian Department of Communications

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

Class B

Federal Communications Commission

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Canadian Department of Communications

This Class B digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe B respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

Compliance to EU Directives

Readers in the European Union (EU) must refer to the Manufacturer's Declaration of Conformity (DoC) for information** pertaining to the CE Mark compliance scheme. The Manufacturer includes a DoC for most every hardware product except for those bought for OEMs, if also available from an original manufacturer that also markets in the EU, or where compliance is not required as for electrically benign apparatus or cables.

To obtain the DoC for this product, click **Declaration of Conformity** at `ni.com/hardref.nsf/`. This website lists the DoCs by product family. Select the appropriate product family, followed by your product, and a link to the DoC appears in Adobe Acrobat format. Click the Acrobat icon to download or read the DoC.

* Certain exemptions may apply in the USA, see FCC Rules §15.103 **Exempted devices**, and §15.105(c). Also available in sections of CFR 47.

** The CE Mark Declaration of Conformity will contain important supplementary information and instructions for the user or installer.

Contents

About This Manual

Chapter 1
NI-CAN Data Types

Chapter 2
NI-CAN Functions

Chapter 3
NI-CAN Objects

Chapter 4
RTSI Programming

Appendix A
NI-CAN Object States

Appendix B
NI-CAN Status

Appendix C
Technical Support Resources

Glossary

Index

Figures

Tables

About This Manual

This manual is a programming reference for functions, objects, and data types in the NI-CAN software. This manual assumes that you are already familiar with the operating system you are using.

How to Use the Manual Set

Use the *Installation Guide, CAN Hardware and the NI-CAN Software for Windows 2000/NT/XP/Me/9*x in the jewel case of your program CD to install and configure your CAN hardware and the NI-CAN software.

Use the *NI-CAN User Manual* to learn the basics of NI-CAN and how to develop an application. The user manual also contains debugging information and examples.

Use this *NI-CAN Programmer Reference Manual* for specific information about each NI-CAN function and object, such as format, parameters, and possible errors.

Conventions Used in This Manual

The following conventions appear in this manual:

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

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

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

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

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

Related Documentation

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

- ANSI/ISO Standard 11898-1993, *Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*
- ANSI/ISO Standard 11519-1, 2 *Road Vehicles—Low Speed Serial Data Communications*, Part 1 and 2
- *CAN Specification Version 2.0*, 1991, Robert Bosch GmbH., Postfach 106050, D-70049 Stuttgart 1
- LabVIEW Online Reference
- Win32 Software Development Kit (SDK) online help

1

NI-CAN Data Types

This chapter describes the data types used by NI-CAN functions and objects.

All data types are given specific names for reference within this manual. In general, all NI-CAN data types begin with NCTYPE_.

Table 1-1. NI-CAN Data Types

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
NCTYPE_*type*_P	NCTYPE_*type* *	N/A	Location of variable with type *type*.
NCTYPE_INT8	signed char	I8	8-bit signed integer.
NCTYPE_INT16	signed short	I16	16-bit signed integer.
NCTYPE_INT32	signed long	I32	32-bit signed integer.
NCTYPE_UINT8	unsigned char	U8	8-bit unsigned integer.
NCTYPE_UINT16	unsigned short	U16	16-bit unsigned integer.
NCTYPE_UINT32	unsigned long	U32	32-bit unsigned integer.
NCTYPE_BOOL	NCTYPE_UINT8	TF (Boolean)	Boolean value. In ANSI C, constants NC_TRUE (1) and NC_FALSE (0) are used for comparisons.
NCTYPE_STRING	char *, array of characters terminated by null character \0	abc (string)	ASCII character string.
NCTYPE_ANY_P	void *	N/A	Reference to variable of unknown type, used in cases where actual data type may vary depending on particular context.

Table 1-1. NI-CAN Data Types (Continued)

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
NCTYPE_OBJH	NCTYPE_UINT32	Type definition ObjHandle (U32)	Handle referring to object.
NCTYPE_VERSION	NCTYPE_UINT32	U32	Version number. Major, minor, subminor, and beta version numbers are encoded in unsigned 32-bit integer from high byte to low byte. Version 2.0 would be hexadecimal 02000000, and Beta version 1.4.2 beta 7 would be hex 01040207.
NCTYPE_DURATION	NCTYPE_UINT32	U32	Time duration indicating elapsed time between two events. Time is expressed in 1 ms increments. 10 seconds is 10000. Special constant NC_DURATION_NONE (0) is used for zero duration, and NC_DURATION_INFINITE (FFFFFFFF hex) is used for infinite duration.
NCTYPE_ABS_TIME	unsigned 64-bit integer compatible with the Win32 FILETIME type	64-bit double-precision floating-point (DBL) compatible with LabVIEW time	For information on use, refer to ncRead function description in Chapter 2, *NI-CAN Functions*.
NCTYPE_ATTRID	NCTYPE_UINT32	U32	Attribute identifier.
NCTYPE_OPCODE	NCTYPE_UINT32	U32	Operation code used with ncAction function.

Table 1-1. NI-CAN Data Types (Continued)

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
NCTYPE_PROTOCOL	NCTYPE_UINT32	U32	Supported device network protocol, such as NC_PROTOCOL_CAN (1).
NCTYPE_BAUD_RATE	NCTYPE_UINT32	U32	Baud rate. 125 kb/s would be encoded as 125000.
NCTYPE_STATE	NCTYPE_UINT32	U32	Object states, encoded as 32-bit mask (one bit for each state). For information, refer to Appendix A, *NI-CAN Object States*.
NCTYPE_STATUS	NCTYPE_INT32	I32	Status returned from all NI-CAN functions. Status is zero for success, less than zero for an error, and greater than zero for a warning. For information, refer to Appendix B, *NI-CAN Status*.
NCTYPE_CAN_ARBID	NCTYPE_UINT32	U32	CAN arbitration ID. 30th bit is accessed using bitmask NC_FL_CAN_ARBID_XTD (20000000 hex). If this bit is clear, CAN arbitration ID is standard (11-bit). If this bit is set, CAN arbitration ID is extended (29-bit). Special constant NC_CAN_ARBID_NONE (CFFFFFFF hex) indicates no CAN arbitration ID.

Table 1-1. NI-CAN Data Types (Continued)

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
NCTYPE_CAN_FRAME	struct	Input terminals of ncWriteNet.vi	Structure used with ncWrite and CAN Network Interface Object. For information, refer to description of CAN Network Interface Object in Chapter 3, *NI-CAN Objects*.
NCTYPE_CAN_STRUCT	struct	Output terminals of ncReadNet.vi and ncReadNetMult.vi	Structure used with ncRead and ncReadMult with the CAN Network Interface Object. For information, refer to description of CAN Network Interface Object in Chapter 3, *NI-CAN Objects*.
NCTYPE_CAN_DATA	struct	Input terminals of ncWriteObj.vi	Structure used with ncWrite and CAN Object. For information, refer to description of CAN Object in Chapter 3, *NI-CAN Objects*.
NCTYPE_CAN_DATA_TIMED	struct	Output terminals of ncReadObj.vi and ncReadObjMult.vi	Structure used with ncRead and ncReadMult with the CAN Object. For information, refer to description of CAN Object in Chapter 3, *NI-CAN Objects*.

2

NI-CAN Functions

This chapter lists the NI-CAN functions and describes the format, purpose, parameters, and return status for each function.

Unless otherwise stated, each NI-CAN function suspends execution of the calling process until it completes.

For information on the status code returned by each NI-CAN function, refer to Appendix B, *NI-CAN Status*.

Function Names

The functions in this chapter are listed alphabetically.

Purpose

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

Format

The format section describes the format of each function for LabVIEW, and for the C programming language.

Input and Output

The input and output parameters for each function are listed.

Description

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

CAN Network Interface Object

The CAN Network Interface Object section gives details about using the function with the CAN Network Interface Object.

CAN Object

The CAN Object section gives details about using the function with the CAN Object.

Examples

Each function description includes sample C language code showing how to use the function. For more detailed examples or for example LabVIEW code, refer to the example programs that are included with your NI-CAN software. For more information on the examples, refer to the *NI-CAN User Manual*.

List of NI-CAN Functions

The following table contains an alphabetical list of the NI-CAN functions.

Table 2-1. NI-CAN Functions

Function	Purpose
`ncAction`	Perform an action on an object.
`ncCloseObject`	Close an object.
`ncConfig`	Configure an object before using it.
`ncCreateNotification`	Create a notification callback for an object (C only).
`ncCreateOccurrence`	Create a notification occurrence for an object (LabVIEW only).
`ncGetAttribute`	Get the value of an object attribute.
`ncGetTimer`	Get the NC_ATTR_ABS_TIME attribute (LabVIEW only).
`ncOpenObject`	Open an object.
`ncRead`	Read the data value of an object.
`ncReadMult`	Read multiple data values from the queue of an object.
`ncReset`	Reset CAN interface.
`ncSetAttribute`	Set the value of an object's attribute.
`ncSetTimer`	Set the NC_ATTR_ABS_TIME attribute (LabVIEW only).
`ncStatusToString`	Convert status code into a descriptive string (C only).
`ncWaitForState`	Wait for one or more states to occur in an object.
`ncWrite`	Write the data value of an object.

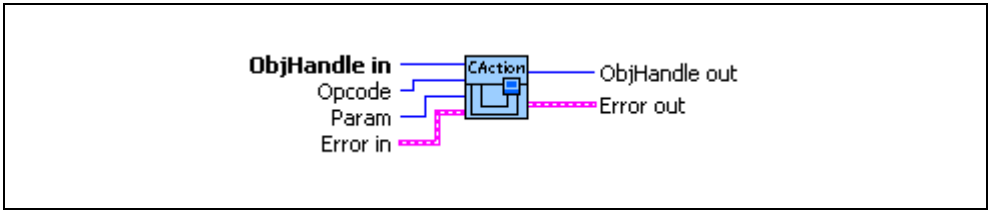
ncAction

Purpose

Perform an action on an object.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncAction(
                          NCTYPE_OBJH ObjHandle,
                          NCTYPE_OPCODE Opcode,
                          NCTYPE_UINT32 Param)
```

Input

`ObjHandle`	Object handle.
`Opcode`	Operation code indicating which action to perform.
`Param`	Parameter whose meaning is defined by `Opcode`.

Description

`ncAction` is a general purpose function you can use to perform an action on the object specified by `ObjHandle`. Its normal use is to start and stop network communication on a CAN Network Interface Object.

For the most frequently used and/or complex actions, NI-CAN provides functions such as `ncOpenObject` and `ncRead`. `ncAction` provides an easy, general purpose way to perform actions that are used less frequently or are relatively simple.

CAN Network Interface Object

NI-CAN propagates all actions on the CAN Network Interface Object up to all open CAN Objects. Table 2-2 describes the actions supported by the CAN Network Interface Object.

Table 2-2. Actions Supported by the CAN Network Interface Object

Opcode	Param	Description
NC_OP_START (80000001 hex)	N/A (ignored)	Transitions network interface from stopped (idle) state to started (running) state. If network interface is already started, this operation has no effect. When a network interface is started, it is communicating on the network. When you execute NC_OP_START on a stopped CAN Network Interface Object, NI-CAN propagates it upward to all open higher-level CAN Objects. Thus, you can use it to start all higher-level network communication simultaneously.
NC_OP_STOP (80000002 hex)	N/A (ignored)	Transitions network interface from started state to stopped state. If network interface is already stopped, this operation has no effect. When a network interface is stopped, it is not communicating on the network. When you execute NC_OP_STOP on a running CAN Network Interface Object, NI-CAN propagates it upward to all open higher-level CAN Objects.
NC_OP_RESET (80000003 hex)	N/A (ignored)	Resets network interface. Stops network interface, then resets the CAN chip to clear the CAN error counters (clear error passive state). Resetting includes clearing all entries from read and write queues. NC_OP_RESET is propagated up to all open higher-level CAN Objects.
NC_OP_RTSI_OUT (80000004 hex)	N/A (ignored)	Output a pulse or toggle on the RTSI line depending upon the NC_ATTR_RTSI_SIG_BEHAV

CAN Object

All actions performed on a CAN Object affect that CAN Object alone, and do not affect other CAN Objects or communication as a whole. To start communications for a CAN Object, you must first start its lower-level CAN Network Interface Object. After starting communications, you can then use `ncAction` to stop and restart an individual CAN Object.

Table 2-3 describes the actions supported by the CAN Object.

Table 2-3. Actions Supported by the CAN Object

Opcode	Param	Description
`NC_OP_START (80000001 hex)`	N/A (ignored)	When `NC_OP_STOP` is used to stop a CAN Object, `NC_OP_START` restarts the CAN Object. This action does not start the CAN Object unless the lower-level CAN Network Interface Object is started (running).
`NC_OP_STOP (80000002 hex)`	N/A (ignored)	Stops the CAN Object. For example, if the CAN Object is configured to transmit data frames periodically, this action stops the periodic transmissions.
`NC_OP_RESET (80000003 hex)`	N/A (ignored)	Resets the CAN Object. Stops the CAN Object, then clears all entries from read and write queues.
`NC_OP_RTSI_OUT (80000004 hex)`	N/A (ignored)	Output a pulse or toggle on the RTSI line depending upon the `NC_ATTR_RTSI_SIG_BEHAV`

Example

This example assumes the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
```

Start communication on a CAN Network Interface Object. Because `Param` is ignored for `NC_OP_START`, you can use any value (this example uses 0).

```
status = ncAction(objh, NC_OP_START, 0);
```

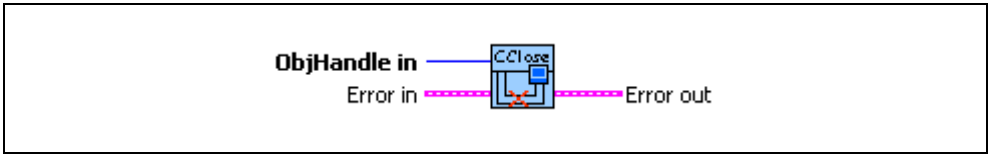
ncCloseObject

Purpose

Close an object.

Format

LabVIEW



C

```
NCTYPE_STATUS      ncCloseObject(NCTYPE_OBJH ObjHandle)
```

Input

`ObjHandle`	Object handle.

Description

`ncCloseObject` closes an object when it no longer needs to be in use, such as when the application is about to exit. When an object is closed, NI-CAN stops all pending operations and clears RTSI configuration for the object, and you can no longer use the `ObjHandle` in your application.

CAN Network Interface Object

`ObjHandle` refers to an open CAN Network Interface Object.

CAN Object

`ObjHandle` refers to an open CAN Object.

Example

This example assumes the following declarations:

```
NCTYPE_STATUS        status;
NCTYPE_OBJH          objh;
```

Close an object.

```
status = ncCloseObject (objh);
```

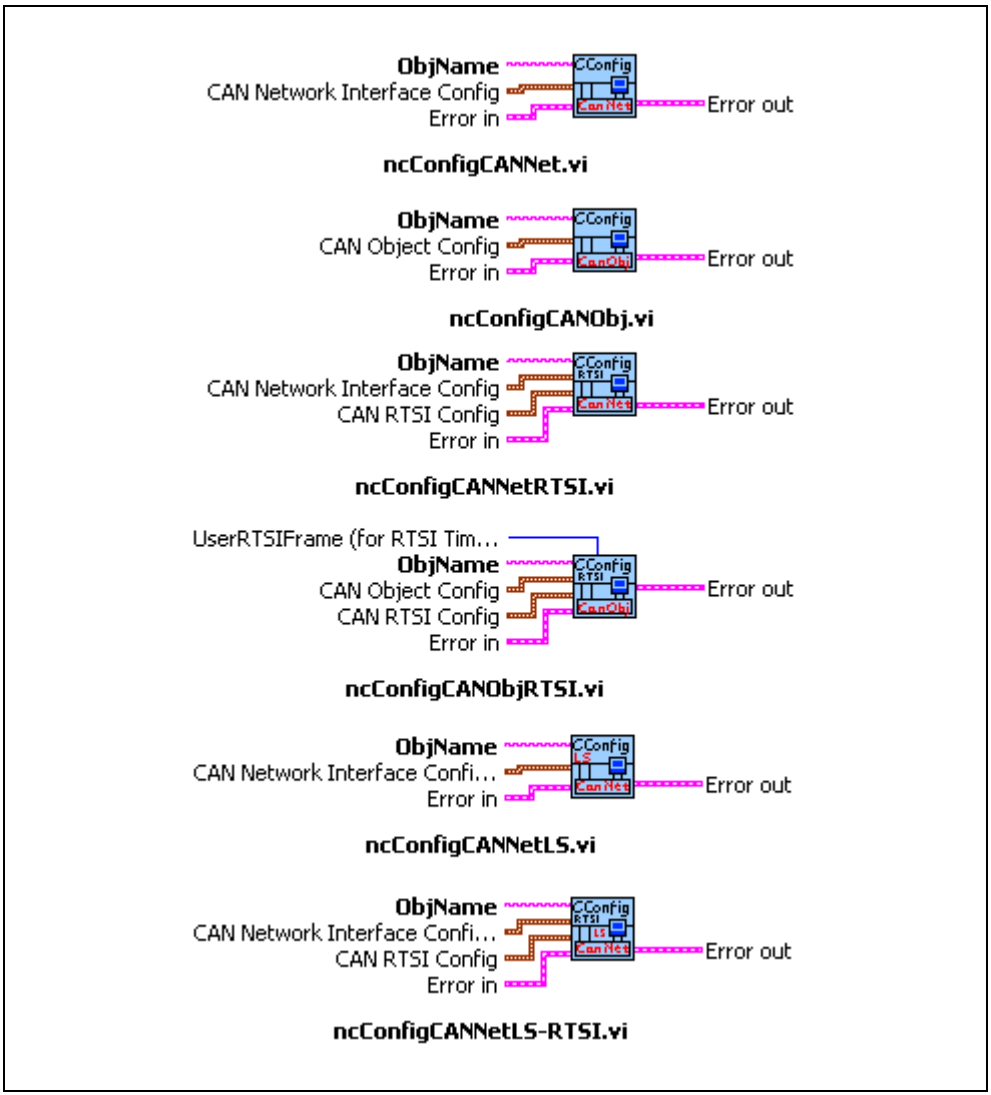
ncConfig

Purpose

Configure an object before using it.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncConfig(
                          NCTYPE_STRING ObjName,
                          NCTYPE_UINT32 NumAttrs,
                          NCTYPE_ATTRID_P AttrIdList,
                          NCTYPE_UINT32_P AttrValueList)
```

Input

ObjName	ASCII name of the object to configure.
NumAttrs	Number of configuration attributes (C only).
AttrIdList	List of configuration attribute identifiers (C only).
AttrValueList	List of configuration attribute values (C only).
ConfigCluster	Cluster of object-specific configuration attribute values (LabVIEW only).
RTSIConfigCluster	Cluster of object-specific RTSI configuration attribute values (LabVIEW only). This is an optimal input.

Description

ncConfig initializes the configuration attributes of an object before opening it. The first NI-CAN function in your application will normally be ncConfig of the CAN Network Interface Object.

ObjName uses the same syntax as ncOpenObject.

NumAttr indicates the number of configuration attributes in AttrIdList and AttrValueList. AttrIdList is an array of attribute IDs, and AttrValueList is an array of values. The attributes in AttrIdList must have Config permissions in the description of the object. The host data type for each value in AttrValueList is NCTYPE_UINT32, which all configuration attributes can use.

Attributes with Config permissions must be initialized prior to opening the object, and cannot be changed using ncSetAttribute.

Using the LabVIEW Configuration Functions

The LabVIEW configuration functions do not require the input parameters AttrIdList and NumAttrs. The configuration attribute values are instead provided in an object-specific cluster. Controls for these configuration clusters can be found in the NI-CAN Controls palette, one for the CAN Network Interface Object (ncNetAttr.ctl), one for the CAN Object (ncObjAttr.ctl), one for RTSI (ncCANRtsiAttr.ctl) and one for DAQ/CAN synchronization configuration (CAN/DAQConfig.ctl). Note that the attribute names in the RTSI configuration cluster are made descriptive to be understood easily.

The ConfigCluster input can be programmed in one of the following ways:

- Place the appropriate control on your front panel, then wire that control into the ConfigCluster input.
- Right-click on the ConfigCluster input and select **Create Control**. This control will not maintain the format, defaults, or description of the original.
- Right-click on the ConfigCluster input and select **Create Constant**. This constant will not maintain the format, defaults, or description of the original.

CAN Network Interface Object

The following are the Config attributes of the CAN Network Interface Object:

NC_ATTR_BAUD_RATE (Baud Rate)
NC_ATTR_START_ON_OPEN (Start on Open)
NC_ATTR_READ_Q_LEN (Read Queue Length)
NC_ATTR_WRITE_Q_LEN (Write Queue Length)
NC_ATTR_CAN_COMP_STD (Standard Comparator)
NC_ATTR_CAN_MASK_STD (Standard Mask)
NC_ATTR_CAN_COMP_XTD (Extended Comparator)
NC_ATTR_CAN_MASK_XTD (Extended Mask)
NC_ATTR_LOG_COMM_ERRS (Low-speed CAN only)
NC_ATTR_RTSI_MODE (RTSI Mode)
NC_ATTR_RTSI_SIGNAL (RTSI Line Selector)
NC_ATTR_RTSI_SIG_BEHAV (RTSI Signal Behavior)
NC_ATTR_RTSI_SKIP (Number of RTSI Triggers to Skip)
NC_ATTR_READ_MULT_SIZE (Size to Be Used with Notifications or Occurrences)

For more information on these configuration attributes, as well as usage of ObjName, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

CAN Object

The following are the Config attributes of the CAN Object:

NC_ATTR_PERIOD (Period)
NC_ATTR_READ_Q_LEN (Read Queue Length)
NC_ATTR_WRITE_Q_LEN (Write Queue Length)
NC_ATTR_RX_CHANGES_ONLY (Receive Changes Only)
NC_ATTR_COMM_TYPE (Communication Type)
NC_ATTR_CAN_TX_RESPONSE (Transmit by Response)
NC_ATTR_CAN_DATA_LENGTH (Data Length)
NC_ATTR_RTSI_MODE (RTSI Mode)
NC_ATTR_RTSI_SIGNAL (RTSI Line Selector)
NC_ATTR_RTSI_SIG_BEHAV (RTSI Signal Behavior)
NC_ATTR_RTSI_SKIP (Number of RTSI Triggers to Skip)

NC_ATTR_READ_MULT_SIZE (Size to Be Used with Notifications or Occurrences)
NC_ATTR_RTSI_FRAME (User-Defined Data Frame)

For more information on these configuration attributes, as well as usage of ObjName, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

Example

This example assumes the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
NCTYPE_ATTRID       AttrIdList[8];
NCTYPE_UINT32       AttrValueList[8];
```

Configure a CAN Network Interface Object.

```
AttrIdList[0] = NC_ATTR_BAUD_RATE;
AttrValueList[0] = 125000;
AttrIdList[1] = NC_ATTR_START_ON_OPEN;
AttrValueList[1] = NC_TRUE;
AttrIdList[2] = NC_ATTR_READ_Q_LEN;
AttrValueList[2] = 10;
AttrIdList[3] = NC_ATTR_WRITE_Q_LEN;
AttrValueList[3] = 10;
AttrIdList[4] = NC_ATTR_CAN_COMP_STD;
AttrValueList[4] = 0;
AttrIdList[5] = NC_ATTR_CAN_MASK_STD;
AttrValueList[5] = 0;
AttrIdList[6] = NC_ATTR_CAN_COMP_XTD;
AttrValueList[6] = 0;
AttrIdList[7] = NC_ATTR_CAN_MASK_XTD;
AttrValueList[7] = 0;
status = ncConfig ("CAN0", 8, AttrIdList, AttrValueList);
```

Configure a CAN Interface Object for low-speed CAN.

All of the above AttrIdList and AttrValueList for Network Interface Object Configuration (as needed) as well as the following:

```
AttrIdList[8] = NC_ATTR_LOG_COMM_ERRS;
AttrValueList[8] = NC_TRUE;
```

Configure RTSI for Network Interface Object.

All of the above `AttrIdList` and `AttrValueList` for Network Interface Object Configuration (as needed) as well as the following:

```
AttrIdList[8] = NC_ATTR_RTSI_MODE;
AttrValueList[8] = NC_RTSI_OUT_ON_TX;
AttrIdList[9] = NC_ATTR_RTSI_SIGNAL;
AttrValueList[9] = 4;
AttrIdList[10] = NC_ATTR_RTSI_SIG_BEHAV;
AttrValueList[10] = NC_RTSI_SIG_PULSE;
AttrIdList[11] = NC_ATTR_RTSI_SKIP;
AttrValueList[11] = 5;
```

ncCreateNotification

Purpose

Create a notification callback for an object (C only).

Format

LabVIEW

N/A (ncCreateOccurrence serves a similar purpose.)

C

```
NCTYPE_STATUS    ncCreateNotification(
                         NCTYPE_OBJH ObjHandle,
                         NCTYPE_STATE DesiredState,
                         NCTYPE_DURATION Timeout,
                         NCTYPE_ANY_P RefData,
                         NCTYPE_NOTIFY_CALLBACK
                         Callback)
```

Input

ObjHandle	Object handle.
DesiredState	States for which notification is sent (see Appendix A, *NI-CAN Object States*).
Timeout	Length of time to wait.
RefData	Pointer to user-specified reference data.
Callback	Address of your callback function.

Description

ncCreateNotification creates a notification callback for the object specified by ObjHandle. The NI-CAN driver uses the notification callback to communicate state changes to your application. The ncCreateNotification function is not applicable to LabVIEW programming. Use the ncCreateOccurrence function to receive notifications within LabVIEW.

This function is normally used when you want to allow other code to execute while waiting for NI-CAN states, especially when the other code does not call NI-CAN functions. If such background execution is not needed, the ncWaitForState function offers better overall performance. The ncWaitForState function cannot be used at the same time as ncCreateNotification.

Upon successful return from ncCreateNotification, the notification callback is invoked whenever one of the states specified by DesiredState occurs in the object. If DesiredState is zero, notifications are disabled for the object specified by ObjHandle.

For information on the bits used for DesiredState, refer to Appendix A, *NI-CAN Object States*.

The NI-CAN driver waits up to Timeout for one of the bits set in DesiredState to become set in the attribute NC_ATTR_STATE. You can use the special Timeout value NC_DURATION_INFINITE to wait indefinitely.

The Callback parameter provides the address of a callback function in your application. Within the Callback function, you can call any of the NI-CAN functions except ncCreateNotification and ncWaitForState.

With the RefData parameter, you provide a pointer that is sent to all notifications for the given object. This pointer normally provides reference data for use within the Callback function. For example, when you create a notification for the NC_ST_READ_AVAIL state, RefData is often the data pointer that you pass to ncRead to read available data. If the callback function does not need reference data, you can set RefData to NULL.

Callback Prototype

```
NCTYPE_STATE        _NCFUNC_ Callback (NCTYPE_OBJH ObjHandle,
                    NCTYPE_STATE State,
                    NCTYPE_STATUS Status,
                    NCTYPE_ANY_P RefData);
```

Callback Parameters

ObjHandle	Object handle.
State	Current state of object (see Appendix A, *NI-CAN Object States*).
Status	Object status.
RefData	Pointer to your reference data.

Callback Return Value

The value you return from the callback indicates the desired states to re-enable for notification. If you no longer want to receive notifications for the callback, return a value of zero.

If you return a state from the callback, and that state is still set in the NC_ATTR_STATE attribute, the callback is invoked again immediately after it returns. For example, if you return NC_ST_READ_AVAIL when the read queue has not been emptied, the callback is invoked again.

Callback Description

In the prototype for Callback, _NCFUNC_ ensures a proper calling scheme between the NI-CAN driver and your callback.

The Callback function executes in a separate thread in your process. Therefore, it has access to any process global data, but not to thread local data. If the callback needs to access global data, you must protect that access using synchronization primitives (such as semaphores), because the callback is running in a different thread context. Alternatively, you can avoid the issue of data protection entirely if the callback simply posts a message to your application using the Win32 PostMessage function. For complete information on multithreading issues, refer to the Win32 Software Development Kit (SDK) online help.

The ObjHandle is the same object handle passed to ncCreateNotification. It identifies the object generating the notification, which is useful when you use the same callback function for notifications from multiple objects.

The State parameter holds the current state(s) of the object that generated the notification (NC_ATTR_STATE attribute). If the Timeout passed to ncCreateNotification expires before the desired states occur, the NI-CAN driver invokes the callback with State equal to zero.

The Status parameter holds the current status of the object. If the notification is sent for the background error and warning states (NC_ST_ERROR or NC_ST_WARNING), Status holds the background status attribute (NC_ATTR_STATUS) of the object. If an error occurs with the notification, State is zero and Status holds the error status. The most common notification error occurs when the Timeout passed to ncCreateNotification expires before the desired states occur (CanErrFunctionTimeout status code). If no background error or warning is reported, and no notification error occurred, Status is CanSuccess.

The RefData parameter is the same pointer passed to ncCreateNotification, and it accesses reference data for the Callback function.

CAN Network Interface Object

The following states apply to the CAN Network Interface Object:

NC_ST_READ_AVAIL	Frame received, available for ncRead.
NC_ST_WRITE_SUCCESS	Frames written with ncWrite were successfully transmitted.
NC_ST_STOPPED	Communication stopped.
NC_ST_ERROR	Error occurred in background.
NC_ST_WARNING	Warning occurred in background.
NC_ST_READ_MULT	Multiple frames available in Read Queue (to be used with ncReadMult).

For more information on these states and the bit values used for each, refer to Appendix A, *NI-CAN Object States*.

CAN Object

The following states apply to the CAN Object:

NC_ST_READ_AVAIL	Data received, available for ncRead.
NC_ST_WRITE_SUCCESS	Data or remote frames written with ncWrite were successfully transmitted.
NC_ST_STOPPED	CAN Object behavior stopped.
NC_ST_ERROR	Error occurred in background.
NC_ST_WARNING	Warning occurred in background.
NC_ST_READ_MULT	Multiple frames available in Read Queue (to be used with ncReadMult).

For more information on these states and the bit values used for each, refer to Appendix A, *NI-CAN Object States*.

Example

Create a notification for the NC_ST_READ_AVAIL state.

```
NCTYPE_STATE        _NCFUNC_ MyCallback (NCTYPE_OBJH ObjHandle,
                    NCTYPE_STATE State,
                    NCTYPE_STATUS Status,
                    NCTYPE_ANY_P RefData){
.
.
.
}

void main()         {
NCTYPE_STATUS           status;
NCTYPE_OBJH             objh
.
.
.
/* Create notification to handle data available in read queue. The
notification waits indefinitely. No RefData is used.*/

status = ncCreateNotification (objh, NC_ST_READ_AVAIL,
NC_DURATION_INFINITE, NULL, MyCallback);
.
.
.
}
```

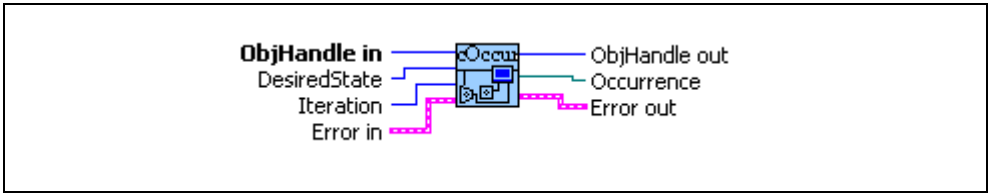
ncCreateOccurrence

Purpose

Create a notification occurrence for an object (LabVIEW only).

Format

LabVIEW



C

N/A (`ncCreateNotification` serves a similar purpose).

Input

`ObjHandle`	Object handle.
`DesiredState`	States for which notification is sent (see Appendix A, *NI-CAN Object States*).
`Iteration`	Loop iteration count (optional).

Output

`Occurrence`	Occurrence that can be used with LabVIEW **Wait on Occurrence** VI.

Description

`ncCreateOccurrence` creates a notification occurrence for the object specified by `ObjHandle`. The NI-CAN driver uses the occurrence callback to communicate state changes to your application. The `ncCreateOccurrence` function is not applicable to C programming. Use the `ncCreateNotification` function to receive notifications within C.

This function is normally used when you want to allow other code to execute while waiting for NI-CAN states, especially when the other code does not call NI-CAN functions. If such background execution is not needed, the `ncWaitForState` function offers better overall performance. The `ncWaitForState` function cannot be used at the same time as `ncCreateOccurrence`.

Upon successful return from ncCreateOccurrence, the notification occurrence is invoked whenever one of the states specified by DesiredState occurs in the object. If DesiredState is zero, notifications are disabled for the object specified by ObjHandle.

For information on the bits used for DesiredState, refer to Appendix A, *NI-CAN Object States*.

If ncCreateOccurrence is called inside a loop, the iteration count of that loop can be wired to the iteration parameter to ensure that the occurrence is only created once. If iteration is left unwired, the occurrence is created each time ncCreateOccurence is called.

The Occurrence output is normally wired into the LabVIEW **Wait on Occurrence** VI. **Wait on Occurrence** takes the Occurrence, and also a timeout and flag indicating whether to ignore a pending state. For more information on **Wait On Occurrence**, refer to the LabVIEW Online Reference.

When **Wait on Occurrence** completes, you should execute code to handle the DesiredState. For example:

1. If DesiredState is NC_ST_READ_AVAIL, you should call ncRead to read the available data.
2. If DesiredState is NC_ST_READ_MULT, you should call ncReadMult to read the available data.

After it has been created, the Occurrence will be set each time a DesiredState goes from false to true. When you no longer want to wait on the Occurrence (for example, when terminating your application), call ncCreateOccurrence with DesiredState zero.

CAN Network Interface Object

The following states apply to the CAN Network Interface Object:

NC_ST_READ_AVAIL	Frame received, available for ncRead.
NC_ST_WRITE_SUCCESS	Frames written with ncWrite were successfully transmitted.
NC_ST_STOPPED	Communication stopped.
NC_ST_ERROR	Error occurred in background.
NC_ST_WARNING	Warning occurred in background.
NC_ST_READ_MULT	Multiple frames available in Read Queue for ncReadMult.

For more information on these states and the bit values used for each, refer to Appendix A, *NI-CAN Object States*.

CAN Object

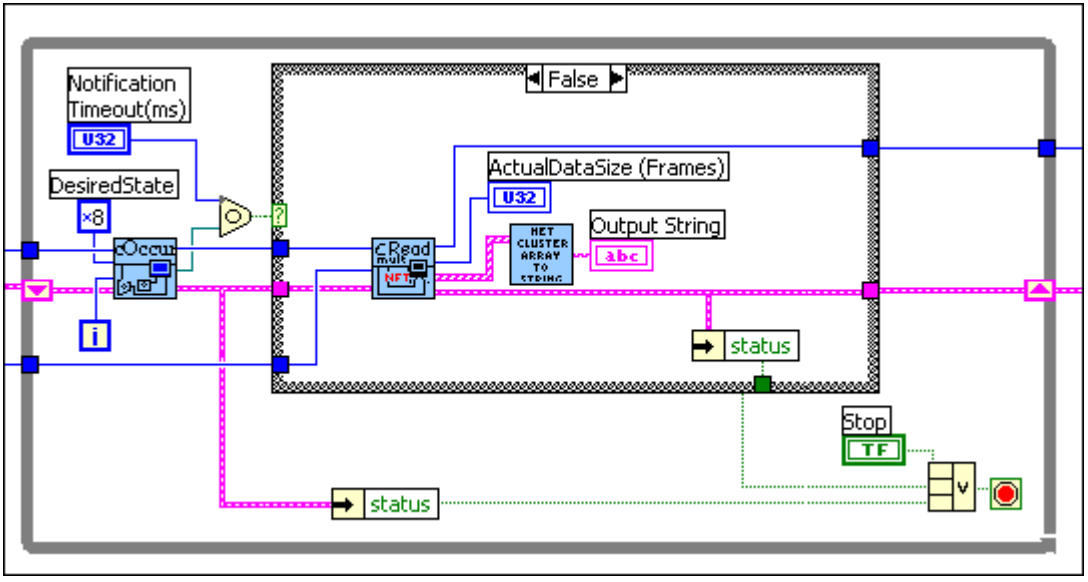
The following states apply to the CAN Object:

NC_ST_READ_AVAIL	Data received, available for ncRead.
NC_ST_WRITE_SUCCESS	Data or remote frames written with ncWrite were successfully transmitted.
NC_ST_STOPPED	CAN Object behavior stopped.
NC_ST_ERROR	Error occurred in background.
NC_ST_WARNING	Warning occurred in background.
NC_ST_READ_MULT	Multiple frames available in Read Queue for ncReadMult.

For more information on these states and the bit values used for each, refer to Appendix A, *NI-CAN Object States*.

Example

Create an occurrence that sets when data is available to be read. Inside a loop, read a CAN frame whenever the occurrence sets. If the occurrence times out after 10 seconds, break from the loop.



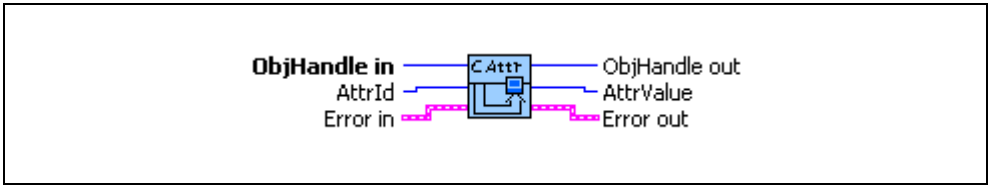
ncGetAttribute

Purpose

Get the value of an object attribute.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncGetAttribute(
                                NCTYPE_OBJH ObjHandle,
                                NCTYPE_ATTRID AttrId,
                                NCTYPE_UINT32 AttrSize,
                                NCTYPE_ANY_P AttrPtr)
```

Input

`ObjHandle`	Object handle.
`AttrId`	Identifier of the attribute to get.
`AttrSize`	Size of the attribute in bytes (C only).

Output

`AttrPtr (AttrValue)`	Returned attribute value. For C, the attribute value is returned to you using the pointer `AttrPtr`. For LabVIEW, the attribute value is returned to you in `AttrValue`.

Description

`ncGetAttribute` gets the value of the attribute specified by `AttrId` from the object specified by `ObjHandle`. Within NI-CAN objects, you use attributes to access configuration settings, status, and other information about the object, but not data.

For C, `AttrPtr` points to the variable used to receive the attribute value. Its type is undefined so that you can use the appropriate host data type for `AttrId`. `AttrSize` indicates the size of the variable that `AttrPtr` points to.

For LabVIEW, this function gets the value of an object's attribute into a LabVIEW `U32` (`AttrValue`), so a size is not needed.

CAN Network Interface Object

For information on the attributes of the CAN Network Interface Object, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

CAN Object

For information on the attributes of the CAN Object, refer to the *CAN Object* section of Chapter 3, *NI-CAN Objects*.

Example

This example assumes the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
NCTYPE_BAUD_RATE    baudrate;
```

Get the value of an object's baud rate attribute.

```
status = ncGetAttribute(objh, NC_ATTR_BAUD_RATE,
sizeof(baudrate), &baudrate);
```

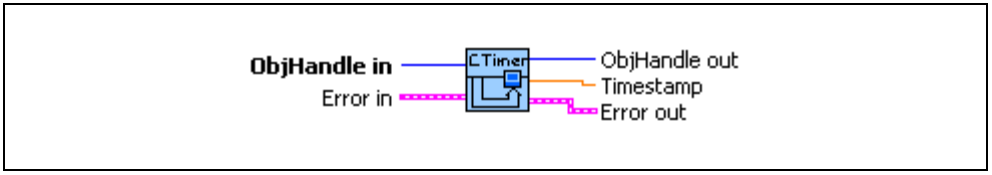
ncGetTimer

Purpose

Get the NC_ATTR_ABS_TIME attribute (LabVIEW only).

Format

LabVIEW



C

N/A (use ncGetAttribute)

Input

ObjHandle	Network interface object handle.

Output

Timestamp	Timestamp attribute value.

Description

Most NI-CAN attributes are of the type U32 in LabVIEW, but since the NC_ATTR_ABS_TIME attribute is of type DBL, a special VI is needed to get the value.

For C, the function ncGetAttribute can get any data type, so it can be used to get NC_ATTR_ABS_TIME.

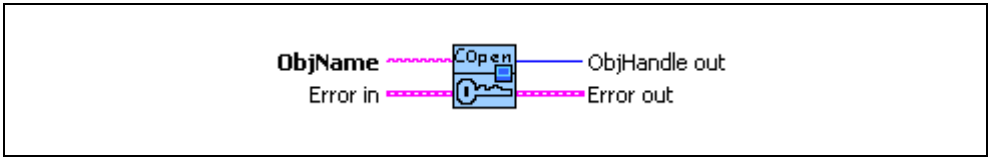
ncOpenObject

Purpose

Open an object.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncOpenObject(
                              NCTYPE_STRING ObjName,
                              NCTYPE_OBJH_P ObjHandlePtr)
```

Input

`ObjName`	ASCII name of the object to open.

Output

`ObjHandlePtr` (`ObjHandle out`)	Object handle you use with all subsequent NI-CAN function calls. For C, the object handle is returned to you using the pointer `ObjHandlePtr`. For LabVIEW, the object handle is returned to you in `ObjHandle out`.

Description

`ncOpenObject` takes the name of an object to open and returns a handle to that object that you use with subsequent NI-CAN function calls.

The `ObjName` syntax specifies the complete hierarchy of an object so that NI-CAN knows both which object to open and where that object is located. This syntax consists of a list of one or more objects in the NI-CAN object hierarchy, each separated by a double colon. When more than one object is required, any number of blanks can exist before or after the double colon.

Specify objects in the NI-CAN hierarchy using a class name followed by an instance number. The class name is a string of letters that describes the class to which the object belongs. Class names are not case-sensitive. The instance number is a numeric value that indicates which object of a class is being specified. Instance numbers are normally specified in decimal

notation. If hexadecimal notation is desired, the number must be preceded by "0x," as in the C programming language. For more information on NI-CAN object names, refer to Chapter 3, *NI-CAN Objects*.

Although NI-CAN can generally be used by multiple applications simultaneously, it does not allow more than one application to open the same object. For example, if one application opens `CAN0`, and another application attempts to open `CAN0`, the second `ncOpenObject` returns the error `CanErrAlreadyOpen`. It is legal for one application to open `CAN0::STD14` and another application to open `CAN0::STD21`, because the two objects are considered distinct.

If `ncOpenObject` is successful, a handle to the newly opened object is returned. You use this object handle for all subsequent function calls for the object.

CAN Network Interface Object

For information on the `ObjName` of the CAN Network Interface Object, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

CAN Object

For information on the `ObjName` of the CAN Object, refer to the *CAN Object* section of Chapter 3, *NI-CAN Objects*.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
```

1. Open a CAN Network Interface Object.

   ```
   status = ncOpenObject ("CAN0", &objh);
   ```

2. Open a CAN Object at standard arbitration ID 14 on `CAN1`.

   ```
   status = ncOpenObject ("CAN1::STD14", &objh);
   ```

3. Open CAN object at extended arbitration ID 2043 hex on `CAN2`.

   ```
   status = ncOpenObject ("CAN2::XTD0x2043", &objh);
   ```

4. This call returns an error of `CanErrBadNameSyntax` because the Z makes the CAN Object name invalid.

   ```
   status = ncOpenObject ("CAN0::ZTD5", &objh);
   ```

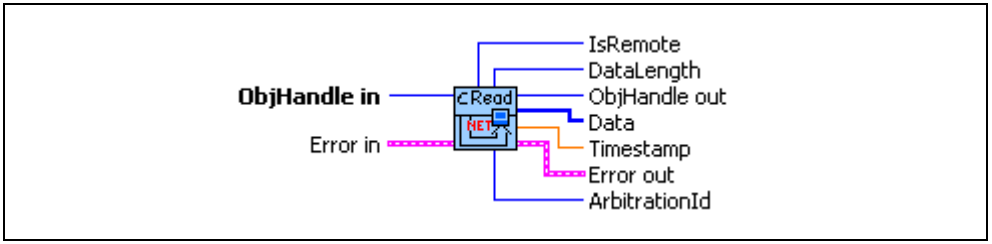
ncRead

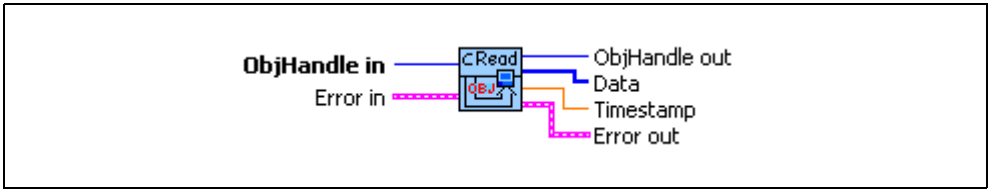
Purpose

Read the data value of an object.

Format

LabVIEW





C

```
NCTYPE_STATUS    ncRead(
                        NCTYPE_OBJH ObjHandle,
                        NCTYPE_UINT32 DataSize,
                        NCTYPE_ANY_P DataPtr)
```

Input

`ObjHandle`	Object handle.
`DataSize`	Size of the data in bytes (C only).

Output

`DataPtr`	Data read from object. For C, the data is returned to you using the pointer `DataPtr`. For LabVIEW, the data is returned to you using object-specific output terminals.

Description

ncRead reads the data value of the object specified by ObjHandle.

For C, DataPtr points to the variable that holds the data. Its type is undefined so that you can use the appropriate host data type. DataSize indicates the size of variable pointed to by DataPtr, and is used to verify that the size you have available is compatible with the configured read size for the object.

For LabVIEW, the data is returned to you using object-specific output terminals.

You use ncRead to obtain data from the read queue of an object. Because NI-CAN handles the read queue in the background, this function does not wait for new data to arrive. To ensure that new data is available before calling ncRead, first wait for the NC_ST_READ_AVAIL state. The NC_ST_READ_AVAIL state transitions from false to true when NI-CAN places a new data item into an empty read queue, and remains true until you read the last data item from the queue.

The ncRead function is useful when you need to process one frame at a time. In order to read multiple frames, such as those for bus analyzer applications, use the ncReadMult function.

When you call ncRead for an empty read queue (NC_ST_READ_AVAIL false), the data from the previous call to ncRead is returned to you again, along with the CanWarnOldData warning. If no data item has yet arrived for the read queue, a default data item is returned, which consists of all zeros.

When a new data item arrives for a full queue, NI-CAN discards the item, and the next call to ncRead returns the CanErrOverflowRead error. You can avoid this overflow behavior by setting the read queue length to zero. When a new data item arrives for a zero length queue, it simply overwrites the previous item without indicating an overflow. The NC_ST_READ_AVAIL state and CanWarnOldData warning still behave as usual, but you can ignore them if you only want the most recent data. You can use the NC_ATTR_READ_Q_LEN attribute to configure the read queue length.

The host data type returned from ncRead is different for each NI-CAN object class. This type normally includes data received from the network along with a timestamp of when that data arrived.

For C, the timestamp that ncRead returns is an unsigned 64-bit integer compatible with the Win32 FILETIME type. When data arrives from the network and is placed in the read queue, NI-CAN obtains this timestamp from the absolute time attribute (NC_ATTR_ABS_TIME) of the CAN Network Interface Object. This absolute time is kept in a Coordinated Universal Time (UTC) format, the standard used for global timekeeping (times that are not specific to local time zone considerations). UTC-based time is loosely defined as the current date and time of day in Greenwich, England. Microsoft defines its UTC time (FILETIME) as a 64-bit counter of 100 ns intervals that have elapsed since 12:00 a.m., January 1, 1601. Because the

timestamp returned by ncRead is compatible with FILETIME, you can pass it into the Win32 FileTimeToLocalFileTime function to convert it to your local time zone format, then pass the resulting local time to the Win32 FileTimeToSystemTime function to convert it to the Win32 SYSTEMTIME type (a structure with fields for year, month, day, and so on). For more information on Win32 time types and functions, refer to the Win32 Software Development Kit (SDK) online help.

For LabVIEW, the timestamp that ncRead returns is compatible with the LabVIEW time format. LabVIEW time is a double-precision floating-point number (DBL) representing the number of seconds that have elapsed since 12:00 a.m., Friday, January 1, 1904, Coordinated Universal Time (UTC). You can pass this timestamp into LabVIEW time functions such as Seconds To Date/Time. You can also display the time in a numeric indicator of type DBL by using **Format & Precision** from the front panel to change from Numeric to Time & Date format (set **Seconds Precision** to 3 to display milliseconds). You can also display this timestamp as milliseconds by dividing by 1,000. Microseconds are displayed by dividing by 1,000,000, and so on. For more information, refer to the LabVIEW Online Reference.

CAN Network Interface Object

The host data type you use with ncRead is NCTYPE_CAN_STRUCT. For LabVIEW, each field of NCTYPE_CAN_STRUCT is returned in a terminal of the NI-CAN Read CAN Network Interface Object function (ncReadNet.vi). For C, NCTYPE_CAN_STRUCT is a structure. Table 2-4 describes the fields of NCTYPE_CAN_STRUCT.

Table 2-4. NCTYPE_CAN_STRUCT Field Names

Field Name	Data Type	Description
Timestamp	NCTYPE_ABS_TIME	Holds value of absolute timer (NC_ATTR_ABS_TIME) when frame was received.
ArbitrationId	NCTYPE_CAN_ARBID	CAN arbitration ID received with frame. For more information on how standard and extended arbitration IDs are encoded, refer to Chapter 1, *NI-CAN Data Types*.

Table 2-4. NCTYPE_CAN_STRUCT Field Names (Continued)

Field Name	Data Type	Description
FrameType	NCTYPE_UINT8	Indicates the type of frame obtained from the read queue: 0 (NC_FRMTYPE_DATA) CAN data frame. The fields of the frame are as described. 1 (NC_FRMTYPE_REMOTE) CAN remote frame. Due to the limitations of the Intel 82527 CAN chip, NI-CAN software currently cannot read remote frames from the CAN Network Interface Object (only write). This frame type does not occur for ncRead. 2 (NC_FRMTYPE_COMM_ERR) CAN communications warning or error indicator. This frame indicates a communications problem reported by the CAN chip or the low-speed CAN transceiver. The ArbitrationId field indicates the type of communications problem. This frame type occurs only when you set the NC_ATTR_LOGCOMM_ERRS attribute to NC_TRUE. For more information, refer to the description of the NC_ATTR_LOG_COMM_ERRORS attribute in the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*. 3 (NC_FRMTYPE_RTSI) RTSI Timestamp. This frame indicates when a RTSI input pulse occurred relative to incoming CAN frames. This frame type occurs only when you set the NC_ATTR_RTSI_MODE attribute to NC_RTSI_TIME_ON_IN. For more information, refer to the description of the NC_ATTR_RTSI_MODE attribute in the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.
DataLength	NCTYPE_UINT8	Number of data bytes in frame.
Data	Array of bytes (NCTYPE_UINT8)	This array holds data bytes (8 maximum).

When a CAN frame arrives from over the network, NI-CAN first checks it for handling by an open CAN Object. If no CAN Object applies, NI-CAN filters the arbitration ID of the frame using the appropriate comparator and mask. If the frame is acceptable, NI-CAN places it into an available entry in the read queue of the CAN Network Interface Object.

CAN Object

The host data type you use with `ncRead` is `NCTYPE_CAN_DATA_TIMED`. For LabVIEW, each field of `NCTYPE_CAN_DATA_TIMED` is returned in a terminal of the NI-CAN Read CAN Object function (`ncReadObj.vi`). For C, `NCTYPE_CAN_DATA_TIMED` is a structure. Table 2-5 describes the fields of `NCTYPE_CAN_DATA_TIMED`.

Table 2-5. NCTYPE_CAN_DATA_TIMED Field Names

Field Name	Data Type	Description
`Timestamp`	`NCTYPE_ABS_TIME`	Holds value of absolute timer (`NC_ATTR_ABS_TIME`) when CAN data frame was received.
`Data`	Array of bytes (`NCTYPE_UINT8`)	Data bytes for CAN Object. Available only when CAN Object is configured to receive data. Length of `Data` is preconfigured using `NC_ATTR_CAN_DATA_LENGTH` attribute.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS        status;
NCTYPE_OBJH          objh;
NCTYPE_CAN_STRUCT    rframe;
NCTYPE_CAN_DATA_TIMEDrdata;
```

1. Read from a CAN Network Interface Object.

   ```
   status = ncRead(objh, sizeof(rframe), &rframe);
   ```

2. Read from a CAN Object.

   ```
   status = ncRead(objh, sizeof(rdata), &rdata);
   ```

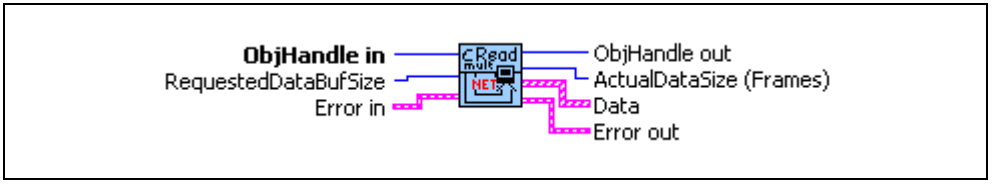
ncReadMult

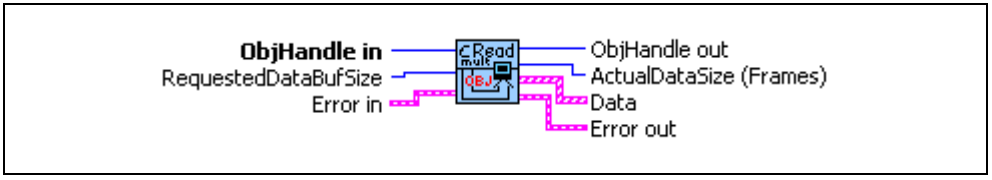
Purpose

Read multiple data values from the queue of an object.

Format

LabVIEW





C

```
NCTYPE_STATUS ncReadMult(
                        NCTYPE_OBJH ObjHandle,
                        NCTYPE_UINT32  DataSize,
                        NCTYPE_ANY_P  DataPtr,
                        NCTYPE_UINT32_P  ActualDataSize);
```

Input

`ObjHandle`	Object handle.
`DataSize`	For C, the size of the data buffer in bytes. For LabVIEW, number of frames desired (not bytes).
`DataPtr`	For C, points to data buffer in which the data returned. For LabVIEW, the data is returned in an array of object-specific clusters.

Output

ActualDataSize	For C, the number of bytes actually returned. For LabVIEW, the number of frames returned (not bytes).

Description

This function returns multiple frames from the read queue of the object specified by ObjHandle. When used with the Network Interface, ncReadMult is useful in analyzer applications where data frames need to be acquired at a high speed and stored for analysis in the future. For single frame and most recent data frame acquisition, you should use ncRead.

For C, DataPtr points to an array of either NCTYPE_CAN_STRUCT or NCTYPE_CAN_DATA_TIMED. DataSize indicates the size of the variable pointed to by DataPtr. This size is specified in bytes in order to verify that the proper data type and alignment is used. When ncReadMult returns, the number of bytes copied into DataPtr is provided in ActualDataSize.

For LabVIEW, RequestedDataBufSize specifies the maximum number of frames that you want to read. The frames are returned in the Data array which contains ActualDataSize frames.

Because NI-CAN handles the read queue in the background, this function does not wait for new data to arrive. To ensure that new data is available before calling ncReadMult, first wait for the NC_ST_READ_MULT state. Refer to Appendix A, *NI-CAN Object States*, for more information on this state.

Unlike the ncRead function, the ncReadMult function does not return the CanWarnOldData warning to indicate zero frames. If there is no new data, the function returns with an ActualDataSize of zero.

The description for CanErrOverflowRead and the host data types is identical to that of ncRead with the exception of CanWarnOldData, described above.

Refer to the ncRead function description for more details on the structures/clusters used with the CAN Network Interface Object and timestamps.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS       Status;
NCTYPE_OBJH         Rxhandle;
```

1. Read from Network Interface Object:

```
NCTYPE_CAN_STRUCTReceiveBuf[140]; //buffer of 140 frames
NCTYPE_UINT32    ActualDataSize;
NCTYPE_CAN_STRUCT_PRecvPtr;
RecvPtr = ReceiveBuf;
Status = ncReadMult(RxHandle, sizeof(ReceiveBuf),RecvPtr,
                &ActualDataSize);
.
.
          //Convert to number of frames
ActualDataSize = ActualDataSize/sizeof(NC_TYPE_CAN_STRUCT);
```

2. Read from CAN Object:

```
NCTYPE_CAN_DATA_TIMEDReceiveBuf[140];//buffer of 140 frames
NCTYPE_UINT32    ActualDataSize;
NCTYPE_CAN_DATA_TIMED_PRecvPtr;
RecvPtr = ReceiveBuf;
Status = ncReadMult(RxHandle, sizeof(ReceiveBuf), RecvPtr,
                &ActualDataSize);
.
.
ActualDataSize = ActualDataSize/sizeof(NCTYPE_CAN_DATA_TIMED);
```

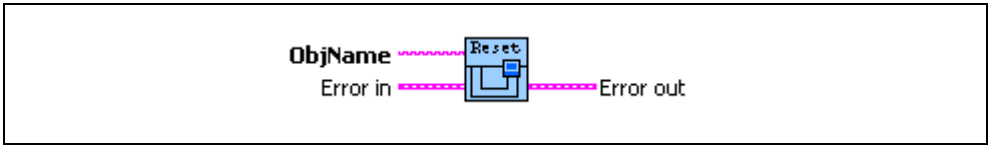
ncReset

Purpose

Reset the CAN interface.

Format

LabVIEW



C

```
NCTYPE_STATUS_NCFUNC_ncReset(
                                NCTYPE_STRING ObjName,
                                NCTYPE_UINT32 Param);
```

Input

`ObjName`	ASCII name of the interface (card) to reset
`Param`	Reserved for future use (set to 0)

Description

This function completely resets the CAN card and ensures that all handles for that card are closed.

The `ObjName` parameter specifies the name of a CAN Network Interface Object, such as `CAN0`. Although `ObjName` refers to only one port, both ports of a 2-port CAN card are reset using `ncReset`.

If an NI-CAN application is terminated prior to closing all handles, the `CanErrNotStopped` or `CanErrAlreadyOpen` error might occur when the application is restarted. This often occurs in LabVIEW when the toolbar **Stop** button is used, or when a wiring problem with `ObjHandle` exists. By making this the first NI-CAN function called in your application (preceding all `ncConfig`), you can avoid problems related to improper termination.

You can only use the `ncReset` function if you plan to run a single NI-CAN application. If you run more than one NI-CAN application, each with `ncReset`, the second `ncReset` call will close all handles for the first application. You should only use the `ncReset` function as a temporary measure. After you update your application so that it successfully closes NI-CAN handles on termination, it should no longer be used.

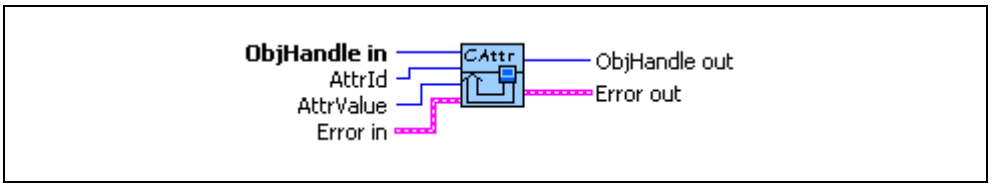
ncSetAttribute

Purpose

Set the value of an object's attribute.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncSetAttribute(
                                NCTYPE_OBJH ObjHandle,
                                NCTYPE_ATTRID AttrId,
                                NCTYPE_UINT32 AttrSize,
                                NCTYPE_ANY_P AttrPtr)
```

Input

`ObjHandle`	Object handle.
`AttrId`	Identifier of the attribute to set.
`AttrSize`	Size of the attribute in bytes (C only).
`AttrPtr (AttrValue)`	New attribute value. For C, you provide the attribute value using the pointer `AttrPtr`. For LabVIEW, you provide the attribute value in `AttrValue`.

Description

`ncSetAttribute` sets the value of the attribute specified by `AttrId` in the object specified by `ObjHandle`. `ncSetAttribute` can be used only for attributes with `Set` permissions, not `Get` (`ncGetAttribute` only) or `Config` (`ncConfig` only).

For C, `AttrPtr` points to the variable that holds the attribute value. Its type is undefined so that you can use the appropriate host data type for `AttrId`. `AttrSize` indicates the size of variable pointed to by `AttrPtr`.

For LabVIEW, this function sets the value of an object's attribute using a LabVIEW `U32` (`AttrValue`), so a size is not needed.

CAN Network Interface Object

For information on the attributes of the CAN Network Interface Object, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

CAN Object

For information on the attributes of the CAN Object, refer to the *CAN Object* section of Chapter 3, *NI-CAN Objects*.

Example

This example assumes the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
NCTYPE_ABS_TIME     abstime;
```

Set the absolute time to zero.

```
abstime.LowPart = 0;
abstime.HighPart = 0;
status = ncSetAttribute(objh, NC_ATTR_ABS_TIME,
sizeof(abstime), &abstime);
```

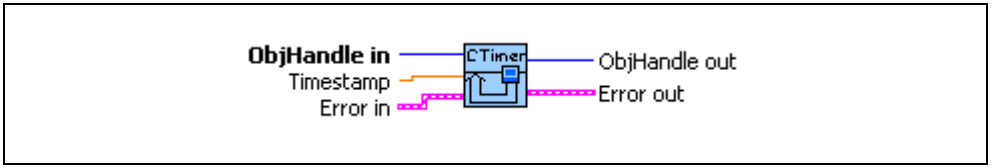
ncSetTimer

Purpose

Set the NC_ATTR_ABS_TIME attribute (LabVIEW only).

Format

LabVIEW



C

N/A (use ncSetAttribute)

Input

ObjHandle	Network Interface object handle.
Timestamp	New timestamp attribute value.

Description

Although most NI-CAN attributes are of type U32 in LabVIEW, the NC_ATTR_ABS_TIME attribute is of type DBL and a special VI is needed to set the value.

For C, the ncSetAttribute function can set any data type, so it can be used to set NC_ATTR_ABS_TIME.

ncStatusToString

Purpose

Convert status code into a descriptive string (C only).

Format

LabVIEW

N/A (use the `Simple Error Handler.vi` in LabVIEW)

C

```
void              ncStatusToString(
                                   NCTYPE_STATUS Status,
                                   NCTYPE_UINT32 SizeofString,
                                   NCTYPE_STRING String)
```

Input

`Status`	Nonzero status code returned from NI-CAN function.
`SizeofString`	Size of `String` buffer (in bytes).

Output

`String`	ASCII string that describes `Status`.

Description

When the status code returned from an NI-CAN function is nonzero, an error or warning is indicated. This function is used to obtain a description of the error/warning for debugging purposes.

The return code is passed into the `Status` parameter. The `SizeofString` parameter indicates the number of bytes available in `String` for the description. The description will be truncated to size `SizeofString` if needed, but a size of 300 characters is large enough to hold any description. The text returned in `String` is null-terminated, so it can be used with ANSI C functions such as `printf`.

For LabVIEW, the standard LabVIEW function `Simple Error Handler.vi`, located in the **Time and Dialog** palette, is used to convert an NI-CAN error cluster into a descriptive string.

For more information, including an example, refer to Appendix B, *NI-CAN Status*.

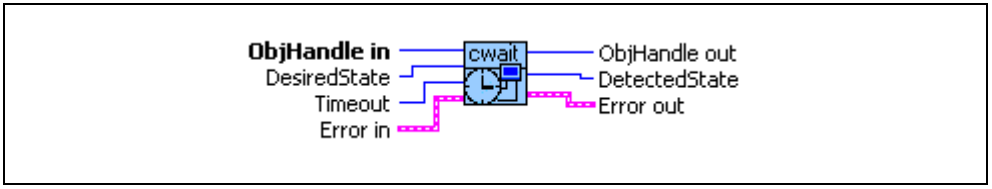
ncWaitForState

Purpose

Wait for one or more states to occur in an object.

Format

LabVIEW



C

```
NCTYPE_STATUS    ncWaitForState(
                                NCTYPE_OBJH ObjHandle,
                                NCTYPE_STATE DesiredState,
                                NCTYPE_DURATION Timeout,
                                NCTYPE_STATE_P StatePtr)
```

Input

`ObjHandle`	Object handle.
`DesiredState`	States to wait for (see Appendix A, *NI-CAN Object States*).
`Timeout`	Length of time to wait.

Output

`StatePtr (DetectedState)`	Current state of object when desired states occur. For C, the state is returned to you using the pointer `StatePtr`. For LabVIEW, the state is returned to you in `DetectedState`.

Description

You use `ncWaitforState` to wait for one or more states to occur in the object specified by `ObjHandle`.

This function waits up to `Timeout` for one of the bits set in `DesiredState` to become set in the attribute `NC_ATTR_STATE`. You can use the special `Timeout` value `NC_DURATION_INFINITE` (`FFFFFFFF` hex) to wait indefinitely.

When the states in DesiredState are detected, the function returns the current value of the NC_ATTR_STATE attribute. If an error occurs, the state returned is zero. For information on the bits used for DesiredState, refer to Appendix A, *NI-CAN Object States*.

While waiting for the desired states, ncWaitForState suspends the current execution. For C, other Win32 threads in your application can still execute. For LabVIEW, functions that are not directly connected to ncWaitForState can execute.

If you want to allow other code in your application to execute while waiting for NI-CAN states, refer to the ncCreateNotification (C only) function.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
NCTYPE_STATE        state;
```

1. Wait no more than 10 seconds for data to arrive in the read queue.

   ```
   status = ncWaitforState(objh, NC_ST_READ_AVAIL, 10000, &state);
   ```

2. Wait no more than 100 milliseconds for a previous ncWrite to succeed, or for a background warning/error, such as bus off, to occur.

   ```
   status = ncWaitforState(objh, (NC_ST_WRITE_SUCCESS | NC_ST_WARNING
   | NC_ST_ERROR), 100, &state);
   ```

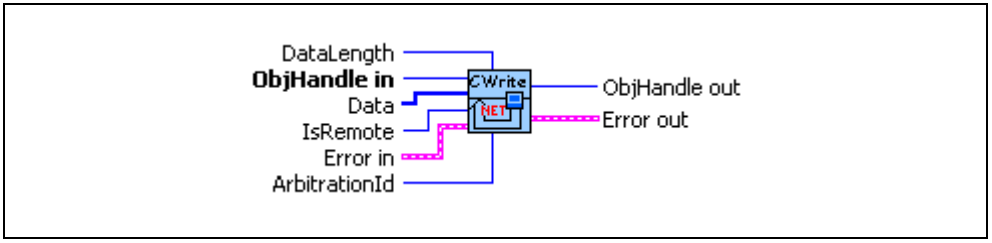
ncWrite

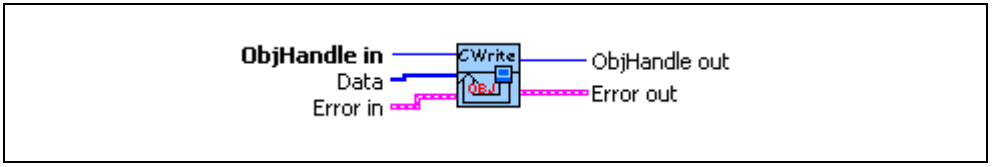
Purpose

Write the data value of an object.

Format

LabVIEW





C

```
NCTYPE_STATUS    ncWrite(
                         NCTYPE_OBJH ObjHandle,
                         NCTYPE_UINT32 DataSize,
                         NCTYPE_ANY_P DataPtr)
```

Input

`ObjHandle`	Object handle.
`DataSize`	Size of the data in bytes.
`DataPtr`	Data written to the object. For C, you provide the data using the pointer `DataPtr`. For LabVIEW, you provide the data using object-specific input terminals.

Description

ncWrite writes the data value of the object specified by ObjHandle.

For C, DataPtr points to the variable from which the data is written. Its type is undefined so that you can use the appropriate host data type. DataSize indicates the size of variable pointed to by DataPtr, and is used to verify that the size you provide is compatible with the configured write size for the object.

For LabVIEW, you provide the data using object-specific input terminals.

You use ncWrite to place data into the write queue of an object. Because NI-CAN handles the write queue in the background, this function does not wait for data to be transmitted on the network. To make sure that the data is transmitted successfully after calling ncWrite, wait for the NC_ST_WRITE_SUCCESS state. The NC_ST_WRITE_SUCCESS state transitions from false to true when the write queue is empty, and NI-CAN has successfully transmitted the last data item onto the network. The NC_ST_WRITE_SUCCESS state remains true until you write another data item into the write queue.

When you configure an object to transmit data onto the network periodically, it obtains data from the object's write queue each period. If the write queue is empty, NI-CAN transmits the data of the previous period again. NI-CAN transmits this data repetitively until the next call to ncWrite.

If an object's write queue is full, a call to ncWrite returns the CanErrOverflowWrite error and NI-CAN discards the data you provide. One way to avoid this overflow error is to set the write queue length to zero. When ncWrite is called for a zero length queue, the data item you provide with ncWrite simply overwrites the previous data item without indicating an overflow. A zero length write queue is often useful when an object is configured to transmit data onto the network periodically, and you simply want to transmit the most recent data value each period. It is also useful when you plan to always wait for NC_ST_WRITE_SUCCESS after every call to ncWrite. You can use the NC_ATTR_WRITE_Q_LEN attribute to configure the write queue length.

The host data type you provide to ncWrite is different for each NI-CAN object class.

CAN Network Interface Object

The host data type you use with ncWrite is NCTYPE_CAN_FRAME. For LabVIEW, each field of NCTYPE_CAN_FRAME is provided in a terminal of the NI-CAN Write CAN Network Interface Object function (ncWriteNet.vi). For C, NCTYPE_CAN_FRAME is a structure. Table 2-6 describes the fields of NCTYPE_CAN_FRAME.

Table 2-6. NCTYPE_CAN_FRAME Field Names

Field Name	Data Type	Description
ArbitrationId	NCTYPE_CAN_ARBID	CAN arbitration ID to transmit with frame. For information on how standard and extended arbitration IDs are encoded, refer to Chapter 1, *NI-CAN Data Types*.
IsRemote	NCTYPE_BOOL	Indicates whether frame is CAN remote frame (NC_TRUE) or CAN data frame (NC_FALSE).
DataLength	NCTYPE_UINT8	When IsRemote is false, this field specifies number of data bytes in frame. When IsRemote is true, it specifies desired number of data bytes.
Data	Array of bytes (NCTYPE_UINT8)	When IsRemote is false, this array holds data bytes (8 maximum).

Sporadic, recoverable errors on the CAN network interface are handled automatically by the protocol, and are not reported as errors from NI-CAN. As such, after ncWrite returns successfully, NI-CAN eventually transmits the frame on the CAN network unless the CanWarnComm warning occurs.

CAN Object

The host data type you use with ncWrite is NCTYPE_CAN_DATA. For LabVIEW, each field of NCTYPE_CAN_DATA is provided in a terminal of the NI-CAN Write CAN Object function (ncWriteObj.vi). For C, NCTYPE_CAN_DATA is a structure.

For CAN Objects configured to transmit a CAN remote frame when you call ncWrite (Receive by Call Using Remote), you do not provide data to ncWrite. For C, you set DataSize to zero. For LabVIEW, you leave the Data terminal of ncWriteObj.vi unconnected. For more information on Receive Value with Call, refer to the description of the *NC_ATTR_COMM_TYPE (Communication Type)* attribute, in Chapter 3.

Table 2-7 describes the field of NCTYPE_CAN_DATA.

Table 2-7. NCTYPE_CAN_DATA Field Name

Field Name	Data Type	Description
Data	Array of bytes (NCTYPE_UINT8)	Data bytes for CAN Object. Available only when CAN Object is configured to transmit data. Length of Data is preconfigured using NC_ATTR_CAN_DATA_LENGTH attribute.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS       status;
NCTYPE_OBJH         objh;
NCTYPE_CAN_FRAME    wframe;
NCTYPE_CAN_DATA     wdata;
```

1. Write to a CAN Network Interface Object.

   ```
   status = ncWrite(objh, sizeof(wframe), &wframe);
   ```

2. Write to a CAN Object.

   ```
   status = ncWrite(objh, sizeof(wdata), &wdata);
   ```

3

NI-CAN Objects

This chapter lists the syntax of the `ObjName` for each object class, specifies what the object encapsulates, and gives an overview of the major features and uses of each object.

For information on how each NI-CAN function is used with the following object classes, refer to Chapter 2, *NI-CAN Functions*.

Object Names

The objects in this chapter are listed in alphabetical order. For each object class, the syntax of its `ObjName` is discussed.

Encapsulates

Each object description includes a brief summary of what the object encapsulates.

Description

The description section gives an overview of the major features and uses of the object.

Attributes

The attributes section lists and describes the attributes for each object. The attributes are listed in alphabetical order.

For each attribute, the description lists its host data type, its attribute ID, and its permissions. Attribute permissions consist of one of the following:

`Get`	You can get the attribute at any time, but never set it.
`Set, Get`	You can get or set the attribute at any time.
`Config, Get`	You can get the attribute at any time, but you can set it only by using the `ncConfig` function. These attributes are called configuration attributes.

CAN Network Interface Object

Object Name

CAN*x*

The letters CAN indicate the class of the CAN Network Interface Object, and *x* is a decimal number starting at zero that indicates which CAN network interface is being referenced (CAN0, CAN1, and so on). Use the NI-CAN Configuration utility to associate instance numbers with physical network interface ports.

Encapsulates

CAN network interface.

Description

The CAN Network Interface Object encapsulates a physical interface to a CAN network, usually a CAN port on an AT, PCI, PCMCIA, or PXI interface.

The communication facilities of the CAN Network Interface Object basically consist of a read queue and a write queue. You use the ncRead function to read CAN frames from the read queue in the order they arrive. When an incoming frame arrives, the NC_ST_READ_AVAIL state sets, to notify you that one or more CAN frames are in the read queue. You use the ncWrite function to write CAN frames to the write queue. NI-CAN transmits CAN frames from the write queue in the order written. When all CAN frames in the write queue are transmitted successfully, the NC_ST_WRITE_SUCCESS state sets.

You can use the CAN Network Interface Object for communication along with CAN Objects. When one or more CAN Objects are open, the CAN Network Interface Object cannot receive frames that would normally be handled by the CAN Objects. For example, if you open the CAN Object named CAN0::STD5, then the CAN Network Interface Object cannot receive frames with standard arbitration ID 5.

If you choose not to configure the CAN Network Interface Object to start automatically (NC_ATTR_START_ON_OPEN attribute is false), it opens in the stopped state (not communicating). To start network communication for the CAN Network Interface Object and all higher level CAN Objects, call ncAction with NC_OP_START. You might want to do this when you have an application that tests an installed CAN network. In this sort of environment, you would load test patterns (lists of data values) into various write queues, then use NC_OP_START to start the test sequence.

Error Active, Error Passive, and Bus Off States

The CAN communication controller used by NI-CAN network interfaces is the Intel 82527. Although this chip provides no direct means of detecting the error passive state, it can detect when one of its error counters increments above 96. When this occurs, NI-CAN sets the NC_ST_WARNING state in the NC_ATTR_STATE attribute of the CAN Network Interface Object and all of its higher level CAN Objects. The background status attribute (NC_ATTR_STATUS) is set with the CanWarnComm status code.

When the transmit error counter of the Intel 82527 increments above 255, the network interface transfers into the bus off state as dictated by the CAN protocol. The network interface stops communication so that you can correct the defect in the network, such as a malfunctioning cable or device. When bus off occurs, the NC_ST_ERROR and NC_ST_STOPPED states are set in the NC_ATTR_STATE attribute of the CAN Network Interface Object and all of its higher level CAN Objects. The background status attribute (NC_ATTR_STATUS) is set with the CanWarnComm status code.

If no CAN devices are connected to the network interface port, and you attempt to transmit a frame, the CanWarnComm status occurs. This warning occurs because the missing acknowledgment bit increments the transmit error counter until the network interface reaches the error passive state, but bus off state is never reached.

Because the error counters in the CAN chip reflect the status of the CAN network, and not necessarily your CAN application, a given CanWarnComm status code will often remain from one run of your application to the next. If you want to clear the CAN chip's error counters (and the CanWarnComm warning) completely when your application starts, use ncAction of NC_OP_RESET to reset the CAN chip, then use ncAction of NC_OP_START to resume communication.

For more information about low-speed communication error handling, refer to the description of the NC_ATTR_LOG_COMM_ERRS attribute in the *CAN Network Interface Object* section of this chapter.

Attributes

NC_ATTR_ABS_TIME (Absolute Time)

Attribute ID	NC_ATTR_ABS_TIME
Hex Encoding	80000008
Data Type	NC_ATTR_ABS_TIME
Permissions	Set, Get
Description	Absolute time of the network interface. The NI-CAN driver uses this attribute for timestamps returned by ncRead. When the NI-CAN driver first initializes (for example, when the host computer is powered on), it is set to the system time of the host computer, and thus keeps the absolute time since that point. You can set this attribute to zero to keep absolute time from a given point, but then the ncRead timestamp is no longer compatible with Win32 FILETIME or LabVIEW time. For more information, refer to the description of the ncRead function in Chapter 2, *NI-CAN Functions*. This attribute applies to all objects of the CAN network interface hardware product. For example, if an interface board contains two network interface ports, this attribute applies to both CAN Network Interface Objects. For LabVIEW, you cannot use the ncGetAttribute and ncSetAttribute VIs for this attribute. Since the host data type of the attribute is DBL in LabVIEW, you must use special ncGetTimer and ncSetTimer VIs.

NC_ATTR_BAUD_RATE (Baud Rate)

Attribute ID	NC_ATTR_BAUD_RATE
Hex Encoding	80000007
Data Type	NCTYPE_BAUD_RATE
Permissions	Config, Get
Description	Baud rate of the network interface. NI-CAN calculates values for various CAN timing parameters and programs them based on the baud rate. All common baud rates are supported, including 10 kb/s, 100 kb/s, 125 kb/s, 250 kb/s, 500 kb/s, and 1000 kb/s.

NC_ATTR_CAN_COMP_STD (Standard Comparator)

Attribute ID	NC_ATTR_CAN_COMP_STD
Hex Encoding	80010001
Data Type	NCTYPE_CAN_ARBID
Permissions	Config, Get
Description	CAN arbitration ID for the standard frame comparator. This comparator filters all incoming standard (11-bit) CAN frames placed into the read queue. The NC_FL_CAN_ARBID_XTD bit must be clear for any value written to this attribute. For more information, refer to the description of NCTYPE_CAN_ARBID in Chapter 1, *NI-CAN Data Types*. If you intend to use CAN Objects as the sole means of receiving standard CAN frames from the network, you should disable all standard frame reception in the CAN Network Interface Object by setting this attribute to NC_CAN_ARBID_NONE (CFFFFFFF hex). With this setting, the network interface is best able to filter out all incoming standard CAN frames except those handled by the CAN Objects.

NC_ATTR_CAN_COMP_XTD (Extended Comparator)

Attribute ID	NC_ATTR_CAN_COMP_XTD
Hex Encoding	80010003
Data Type	NCTYPE_CAN_ARBID
Permissions	Config, Get
Description	CAN arbitration ID to the extended frame comparator. This comparator filters all incoming extended (29-bit) CAN frames placed into the read queue. The NC_FL_CAN_ARBID_XTD bit must be set for any value written to this attribute. For more information, refer to the description of NCTYPE_CAN_ARBID in Chapter 1, *NI-CAN Data Types*. If you intend to use CAN Objects as the sole means of receiving extended CAN frames from the network, you should disable all extended frame reception in the CAN Network Interface Object by setting this attribute to NC_CAN_ARBID_NONE (CFFFFFFF hex). With this setting, the network interface is best able to filter out all incoming extended CAN frames except those handled by the CAN Objects.

NC_ATTR_CAN_MASK_STD (Standard Mask)

Attribute ID	NC_ATTR_CAN_MASK_STD
Hex Encoding	80010002
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Bitmask used in conjunction with NC_ATTR_CAN_COMP_STD for filtration of incoming standard CAN frames. For each bit set in the mask, NI-CAN checks the corresponding bit in the standard frame comparator for a match. Bits in the mask that are clear are treated as don't-cares. For example, hex 000007FF means to compare all 11 bits of incoming standard CAN frames. If the standard frame comparator is NC_CAN_ARBID_NONE, NI-CAN ignores this mask, because all standard frame reception is disabled in the CAN Network Interface Object.

NC_ATTR_CAN_MASK_XTD (Extended Mask)

Attribute ID	NC_ATTR_CAN_MASK_XTD
Hex Encoding	80010004
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Bitmask used in conjunction with NC_ATTR_CAN_COMP_XTD for filtration of incoming extended CAN frames. For each bit set in the mask, NI-CAN checks the corresponding bit in the extended frame comparator for a match. Bits in the mask that are clear are treated as don't-cares. For example, hex 1FFFFFFF means to compare all 29 bits of incoming extended CAN frames. If the extended frame comparator is NC_CAN_ARBID_NONE, NI-CAN ignores this mask.

NC_ATTR_LOG_COMM_ERRS (Low-Speed CAN)

Attribute ID	NC_ATTR_LOG_COMM_ERRS
Hex Encoding	8001000A
Data Type	NCTYPE_BOOL
Permissions	Config, Get
Description	This attribute only applies to low-speed CAN interfaces. If this CAN Network Interface attribute is set to NC_TRUE (by adding to the ncConfig list), CAN communication errors are logged to the interface read queue, and are not reported in NI-CAN status. When looking at a frame read using ncRead, a CAN communication error is detected by checking from the following special value in the FrameType field of the NCTYPE_CAN_STRUCT structure: #define NC_FRMTYPE_COMM_ERR 2 When FrameType has this value, the TimeStamp field indicates the time when the error occurred (or cleared). The ArbitrationId field will consist of one of the following values: 0x4000000B-0x4006000B Bus off warning occurred (error passive) 0x8000000B-0x8006000B Bus off error occurred (bus off) 0x0000000B Bus off warning/error has cleared 0x8000000C Low-speed transceiver warning detected 0x0000000C Low-speed transceiver warning cleared **Note:** The default value for this attribute is NC_FALSE. For LabVIEW, this attribute is configured using the Network Interface Config Cluster-LS and the ncConfigCANNetLS.vi.

NC_ATTR_PROTOCOL (Protocol)

Attribute ID	NC_ATTR_PROTOCOL
Hex Encoding	80000001
Data Type	NCTYPE_PROTOCOL
Permissions	Get
Description	Protocol implemented by the CAN Network Interface Object. The value is always NC_PROTOCOL_CAN (00000001 hex).

NC_ATTR_PROTOCOL_VERSION (Protocol Version)

Attribute ID	NC_ATTR_PROTOCOL_VERSION
Hex Encoding	80000002
Data Type	NCTYPE_VERSION
Permissions	Get
Description	Version that indicates the level of conformance to the protocol specification. The value is always hex 02000200 (major version 2, minor version 0, subminor B), to indicate conformity with CAN 2.0 Parts A and B. The CAN implementation under NI-CAN also complies with ISO 11898.

NC_ATTR_READ_MULT_SIZE (ReadMult Size for Notification)

Attribute ID	NC_ATTR_READ_MULT_SIZE
Hex Encoding	0x8001000B
Data Type	NCTYPE_UINT32
Permissions	Set, Get
Description	Sets the size (in number of frames) that NI-CAN will use to notify the user when specified amount of frames are available in the Read Queue. Once the application receives the notification, the user can call ncReadMult to read the desired number of frames. By using this functionality, a user does not need to poll the queue constantly to read frames. Notice that the ncCreateNotification (or ncCreateOccurance) function must use a desired state of NC_ST_READ_MULT (hex encoding 0x00000008). In LabVIEW, you can set the attribute by using the ncSetAttribute function after calling ncConfig and ncOpen for the object. The user then needs to call ncCreateOccurance, as shown in this manual, with the aforementioned state (0x08) as the Desired State. In C, you can do this by either adding it to the Network Interface attribute configuration or calling ncSetAttribute function. The function call to ncCreateNotification must have the Desired State of NC_ST_READ_MULT.

NC_ATTR_READ_PENDING (Read Entries Pending)

Attribute ID	NC_ATTR_READ_PENDING
Hex Encoding	80000011
Data Type	NCTYPE_UINT32
Permissions	Get
Description	Indicates the number of pending entries in the read queue. If NC_ATTR_READ_PENDING is zero, the NC_ST_READ_AVAIL state is clear.

NC_ATTR_READ_Q_LEN (Read Queue Length)

Attribute ID	NC_ATTR_READ_Q_LEN
Hex Encoding	80000013
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Length (maximum number of entries) for the read queue. For more information, refer to the description of the ncRead function in Chapter 2, *NI-CAN Functions*.

NC_ATTR_RTSI_MODE (RTSI Mode)

Attribute ID	NC_ATTR_RTSI_MODE
Hex Encoding	0x80000017
Data Type	NCTYPE_UINT32
Permissions	Config, Get

Description	This attribute defines whether the user needs to configure the CAN object as an RTSI driver or the DAQ board as the RTSI driver. The following values can be used: **Attribute values:** **In C** — **In LabVIEW RTSI Config Cluster** NC_RTSI_NONE — Disable RTSI NC_RTSI_TX_ON_IN — On RTSI Input—Transmit CAN Frame NC_RTSI_TIME_ON_IN — On RTSI Input—Timestamp RTSI event NC_RTSI_OUT_ON_RX — RTSI Output on Receiving CAN Frame NC_RTSI_OUT_ON_TX — RTSI Output on Transmitting CAN Frame NC_RTSI_OUT_ACTION_ONLY — RTSI Output on ncAction call Each mode is explained in detail below. **NC_RTSI_TX_ON_IN (On RTSI Input-Transmit CAN Frame):** In this mode, NI-CAN will transmit the most recent frame on an incoming RTSI trigger on the RTSI line configured via NC_ATTR_RTSI_SIGNAL. To begin transmission, a frame must be written to the write queue of the object (by calling ncWrite) and an RTSI signal applied to the configured RTSI line. NI-CAN will retransmit the last frame until a new frame is enqueued. **NC_RTSI_TIME_ON_IN (On RTSI Input—Timestamp RTSI event):** In this mode, NI-CAN will timestamp an incoming RTSI trigger on the RTSI line configured via NC_ATTR_RTSI_SIGNAL and enqueue in the object's read queue a frame containing special entries in the following fields (as noted): Timestamp: Time when RTSI event occurred Arbitration ID: 0x40000001

Description (continued)	FrameType 3 (NC_FRMTYPE_RTSI) DataLength: RTSI line number that produces the event Data[8]: Unchanged **NC_RTSI_OUT_ON_RX (RTSI Output on Receiving CAN Frame):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL (RTSI Line Number) whenever a frame is enqueued in the read queue of that object. **NC_RTSI_OUT_ON_TX (RTSI Output on Transmitting CAN Frame):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL (RTSI Line Number) whenever a frame is successfully transmitted. **NC_RTSI_OUT_ACTION_ONLY (RTSI Output on ncAction call):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL whenever the user calls the ncAction function. With this function, a user can set/toggle an RTSI line high or low.

NC_ATTR_RTSI_SIG_BEHAV (RTSI Behavior)

Attribute ID	NC_ATTR_RTSI_SIG_BEHAV
Hex Encoding	0x80000019
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute is used when a CAN object is used to output RTSI signals and defines whether the RTSI line is pulsed or toggled. **Attribute values:** NC_RTSISIG_PULSE (Output RTSI Pulse): This pulses the RTSI line with a 100 µs pulse. NC_RTSISIG_TOGGLE (Toggle RTSI Line): This toggles the RTSI line. If the previous state was high, it will be toggled low, and vice versa.

NC_ATTR_RTSI_SIGNAL (RTSI Line Number)

Attribute ID	NC_ATTR_RTSI_SIGNAL
Hex Encoding	0x80000018
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute defines the RTSI B signal to be associated with the CAN object. **Attribute values:** 0 to 7 (for RTSI B signal lines) **Note:** In the hardware, four RTSI lines are for input and four lines are for output. Hence, four CAN objects can configure RTSI lines as input and four can configure RTSI lines as output. An error will be reported when these limits are exceeded. **Note:** For low-speed and dual-speed boards, two lines are available as input and three (3) lines are available as output. The unavailable lines are used for low-speed transceiver fault reporting. **Note:** For PXI-CAN cards, RTSI signal 0 is unavailable. There is no limitation on which lines can be used as input or output.

NC_ATTR_RTSI_SKIP (RTSI Skip)

Attribute ID	NC_ATTR_RTSI_SKIP
Hex Encoding	0x80000021
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute defines the number of RTSI events to skip before logging them to the read queue for that object. This attribute is used with NC_ATTR_RTSI_MODE and with an attribute value of NC_RTSI_TIME_ON_IN. **Attribute values:** Any user number.

NC_ATTR_RX_Q_LEN (Intermediate Queue Length)

Attribute ID	NC_ATTR_RX_Q_LEN
Hex Encoding	8001000C
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Sets the length of the onboard receive queue. This queue is a transitionary queue between the CAN Controller and the actual read queue of the CAN Network Interface and/or CAN Object. The default length of this queue is 50. In certain high-traffic situations, you might receive an error status CanErrOverflowRxQueue. If you see this receive queue overflow error, increase the length to the maximum number of frames that you would expect to receive in a burst (back-to-back).

NC_ATTR_SOFTWARE_VERSION (Software Version)

Attribute ID	NC_ATTR_SOFTWARE_VERSION
Hex Encoding	80000003
Data Type	NCTYPE_VERSION
Permissions	Get
Description	Version of the NI-CAN driver that implements this object as well as all objects above it in the object hierarchy. This is the National Instruments version number, not the version of the protocol.

NC_ATTR_START_ON_OPEN (Start On Open)

Attribute ID	NC_ATTR_START_ON_OPEN
Hex Encoding	80000006
Data Type	NCTYPE_BOOL
Permissions	Config, Get
Description	Indicates whether communication starts for the CAN Network Interface Object (and all CAN Objects above it in the hierarchy) immediately after you open an object with ncOpenObject. You must always set this attribute within the NI-CAN Configuration utility. It is normally set to true after you use the utility to specify needed configuration attributes such as baud rate. When this attribute is set to true, NI-CAN starts communication transparently. When this attribute is set to false, you must use ncAction to issue NC_OP_START on the CAN Network Interface Object to begin network communication.

NC_ATTR_STATE (Object State)

Attribute ID	NC_ATTR_STATE
Hex Encoding	80000009
Data Type	NCTYPE_STATE
Permissions	Get
Description	Current state of the CAN network interface. For more information, refer to Appendix A, *NI-CAN Object States*.

NC_ATTR_STATUS (Object Status)

Attribute ID	NC_ATTR_STATUS
Hex Encoding	8000000A
Data Type	NCTYPE_STATUS
Permissions	Get
Description	Background status of the CAN network interface. Unless the NC_ST_WARNING or NC_ST_ERROR states are set in NC_ATTR_STATE, this attribute always returns CanSuccess. When you read an error or warning from this attribute, NI-CAN clears the appropriate state and sets the background status back to CanSuccess. Sporadic, recoverable errors on the CAN network interface are handled automatically by the protocol, and are not reported as errors from NI-CAN. If a background error occurs, you can read it from this attribute, or obtain it from the next call to ncRead or ncWrite.

NC_ATTR_WRITE_PENDING (Write Entries Pending)

Attribute ID	NC_ATTR_WRITE_PENDING
Hex Encoding	80000012
Data Type	NCTYPE_UINT32
Permissions	Get
Description	Indicates the number of pending entries in the write queue. If NC_ST_WRITE_PENDING is zero, the NC_ST_WRITE_SUCCESS state is set (after NI-CAN successfully transmits the final frame).

NC_ATTR_WRITE_Q_LEN (Write Queue Length)

Attribute ID	NC_ATTR_WRITE_Q_LEN
Hex Encoding	80000014
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Length (maximum number of entries) for the write queue. For more information, refer to the description of the ncWrite function in Chapter 2, *NI-CAN Functions*.

CAN Object

Object Name

CAN*x*::STD*Arbitration ID*
CAN*x*::XTD*Arbitration ID*

CAN*x* is the name of a CAN Network Interface Object such as CAN0. The letters STD and XTD indicate the class of the CAN Object, specifying whether it uses a standard (11-bit) arbitration ID or an extended (29-bit) arbitration ID. You normally specify the actual *Arbitration ID* of the CAN Object as a decimal number, but you can use hexadecimal notation by including a “0x” at the beginning of the hexadecimal notation.

Encapsulates

CAN arbitration ID and its associated data.

Description

When a network frame is transmitted on a CAN-based network, it always begins with the arbitration ID. This arbitration ID is primarily used for collision resolution when more than one frame is transmitted simultaneously, but often is also used as a simple mechanism to identify data. The CAN arbitration ID, along with its associated data, is referred to as a CAN Object.

The NI-CAN implementation of CAN provides high-level access to CAN Objects on an individual basis. You can configure each CAN Object for different forms of communication (such as periodic polling, receiving unsolicited CAN data frames, and so on). After you configure a CAN Object and open it for communication, use the ncRead and ncWrite functions to access the data of the CAN Object. The NI-CAN driver performs all other details regarding the object.

Attributes

NC_ATTR_CAN_DATA_LENGTH (Data Length)

Attribute ID	NC_ATTR_CAN_DATA_LENGTH
Hex Encoding	80010007
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	NC_ATTR_CAN_DATA_LENGTH indicates the number of bytes of data contained in CAN data frames for the CAN Object. This number is also placed into the Data Length Code (DLC) field of transmitted CAN data frames or CAN remote frames (although CAN remote frames do not contain actual data bytes).

NC_ATTR_CAN_TX_RESPONSE (Transmit by Response)

Attribute ID	NC_ATTR_CAN_TX_RESPONSE
Hex Encoding	80010006
Data Type	NCTYPE_BOOL
Permissions	Config, Get
Description	The NC_ATTR_CAN_TX_RESPONSE attribute applies only to CAN Object configurations in which the Communication Type (NC_ATTR_COMM_TYPE) is set to Transmit Data by Call, Transmit Data Periodically, or Transmit Periodic Waveform. For those configurations, NC_ATTR_CAN_TX_RESPONSE specifies whether the CAN Object should automatically respond with the previously transmitted CAN data frame when it detects an incoming CAN remote frame. When set to NC_FALSE, the CAN Object transmits CAN data frames only as configured, and ignores all incoming CAN remote frames for its arbitration ID. When set to NC_TRUE, the CAN Object responds to incoming CAN remote frames. CAN data frames transmitted due to incoming CAN remote frames are independent of any CAN data frames transmitted as a result of configured behavior. If you know that a given CAN Object will not receive CAN remote frames, you should set this attribute to NC_FALSE so that NI-CAN can ignore such frames.

NC_ATTR_COMM_TYPE (Communication Type)

Attribute ID	NC_ATTR_COMM_TYPE
Hex Encoding	80000016
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	The NC_ATTR_COMM_TYPE (Communication Type) attribute configures the fundamental behavior of the CAN Object. The values for Communication Type are described in the *Values for Communication Type* section, later in this chapter. Values that Receive are always used to receive CAN data frames (and possibly transmit CAN remote frames). Values that Transmit are always used to transmit CAN data frames (and possibly receive CAN remote frames).

NC_ATTR_PERIOD (Period)

Attribute ID	NC_ATTR_PERIOD
Hex Encoding	8000000F
Data Type	NCTYPE_DURATION
Permissions	Config, Get
Description	When you set the Communication Type (NC_ATTR_COMM_TYPE) to Transmit Data Periodically, Transmit Periodic Waveform, or Receive Periodically Using Remote, this attribute specifies the time in milliseconds between subsequent transmissions. When you set the Communication Type to Receive Unsolicited or Transmit by Response Only, this attribute specifies a watchdog timeout. A watchdog timeout of zero disables the watchdog timer. When you set the Communication Type to Transmit Data by Call or Receive Data By Call Using Remote, this attribute specifies the minimum interval between subsequent transmissions. A minimum interval of zero disables the minimum interval timer.

NC_ATTR_READ_MULT_SIZE

Attribute ID	NC_ATTR_READ_MULT_SIZE
Hex Encoding	0x8001000B
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Sets the size (in number of frames) that NI-CAN will use to notify the user when specified amount of frames are available in the Read Queue. Once the application receives the notification, the user can call ncReadMult to read the desired number of frames. By using this functionality, a user does not need to poll the queue constantly to read frames. Note that the ncCreateNotification (or ncCreateOccurence) function must use a desired state of NC_ST_READ_MULT (hex encoding 0x00000008). In LabVIEW, the attribute can be set by using the ncSetAttribute function after calling ncConfig and ncOpen for the object. The user then needs to call ncCreateOccurence, as shown in this manual, with the aforementioned state (0x08) as the Desired State. In C, you can do this by either adding it to the Network Interface attribute configuration or calling the ncSetAttribute function. The function call to ncCreateNotification must have the Desired State of NC_ST_READ_MULT.

NC_ATTR_READ_PENDING (Read Entries Pending)

Attribute ID	NC_ATTR_READ_PENDING
Hex Encoding	80000011
Data Type	NCTYPE_UINT32
Permissions	Get
Description	Indicates the number of pending entries in the read queue. If NC_ATTR_READ_PENDING is zero, the NC_ST_READ_AVAIL state is clear.

NC_ATTR_READ_Q_LEN (Read Queue Length)

Attribute ID	NC_ATTR_READ_Q_LEN
Hex Encoding	80000013
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Length (maximum number of entries) for the read queue. For more information, refer to the description of the ncRead function in Chapter 2, *NI-CAN Functions*.

NC_ATTR_RTSI_FRAME (User RTSI Frame)

Attribute ID	NC_ATTR_RTSI_FRAME
Hex Encoding	0x80000020
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Use this attribute when a CAN object is to be configured with the attribute NC_ATTR_RTSI_MODE and an attribute value of NC_RTSI_TIME_ON_IN. Because the CAN object's receiving structure contains only the Timestamp and Data[8] fields, you must specify a 4-byte data frame that NI-CAN can use in the first four bytes of the Data[8] field, to help distinguish the RTSI event from other data frames. This user frame is configured via the NC_ATTR_RTSI_FRAME attribute in the RTSI configuration for the CAN object. **Attribute values:** Any user-defined usigned32 number in hex. For example, 0xAABBCCDD. **Note:** In LabVIEW the default configuration is Ox5254349 which is the ASCII value for RTSI.

NC_ATTR_RTSI_MODE (RTSI Mode)

Attribute ID	NC_ATTR_RTSI_MODE
Hex Encoding	0x80000017
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute defines whether the user needs to configure the CAN object as an RTSI driver or the DAQ board as the RTSI driver. The following values can be used: **Attribute values:** **In C** — **In LabVIEW RTSI Config Cluster** NC_RTSI_NONE — Disable RTSI NC_RTSI_TX_ON_IN — On RTSI Input—Transmit CAN Frame NC_RTSI_TIME_ON_IN — On RTSI Input—Timestamp RTSI event NC_RTSI_OUT_ON_RX — RTSI Output on Receiving CAN Frame NC_RTSI_OUT_ON_TX — RTSI Output on Transmitting CAN Frame NC_RTSI_OUT_ACTION_ONLY — RTSI Output on ncAction call Each of the modes is explained in detail below. **NC_RTSI_TX_ON_IN (On RTSI Input-Transmit CAN Frame):** In this mode, NI-CAN will transmit the most recent frame on an incoming RTSI trigger on the RTSI line configured via NC_ATTR_RTSI_SIGNAL. The CAN Object can be configured as: • NC_CAN_COMM_TX_BY_CALL (transmit frame by calling ncWrite) • NC_CAN_COMM_TX_PERIODIC (periodic transmission) • NC_CAN_COMM_TX_WAVEFORM (waveform transmit) In all three configurations, a CAN frame is transmitted on every incoming RTSI trigger. The period is ignored (if nonzero). **Note:** For transmitting waveform, follow the instructions in the *CAN Object* section.

Description (continued)	For an object configured with NC_CAN_COMM_TX_BY_CALL, transmission is begun by writing a frame containing the write queue of the object (by calling ncWrite) and an RTSI signal applied to the configured RTSI line. NI-CAN will retransmit the last frame until a new frame is enqueued. **NC_RTSI_TIME_ON_IN (On RTSI Input—Timestamp RTSI event):** In this mode, NI-CAN will timestamp an incoming RTSI trigger on the RTSI line configured via NC_ATTR_RTSI_SIGNAL and enqueue (in the object's read queue) a frame containing special entries in the following fields (as noted): Timestamp: Time when RTSI event occurred. Data[8]: User-defined frame (first 4 bytes) defined by NC_ATTR_RTSI_FRAME. See description of the NC_ATTR_RTSI_FRAME attribute for more details. **NC_RTSI_OUT_ON_RX (RTSI Output on Receiving CAN Frame):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL (RTSI Line Number) whenever a frame is enqueued in that object's read queue. This RTSI configuration can be used when the Object is configured as: NC_CAN_COMM_RX_UNSOL (Receive Unsolicited) **NC_RTSI_OUT_ON_TX (RTSI Output on Transmitting CAN Frame):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL (RTSI Line Number) whenever a frame is successfully transmitted. You can use this RTSI configuration when the object is configured as: • NC_CAN_COMM_TX_BY_CALL (Transmit Data by call) • NC_CAN_COMM_TX_PERIODIC (Transmit Data Periodically) • NC_CAN_COMM_TX_WAVEFORM (Transmit Periodic Waveform) **NC_RTSI_OUT_ACTION_ONLY (RTSI Output on ncAction call):** In this mode, NI-CAN will output an RTSI trigger on the line configured via NC_ATTR_RTSI_SIGNAL whenever the user calls the ncAction function. With this function, a user can set/toggle an RTSI line high or low.

NC_ATTR_RTSI_SIG_BEHAV (RTSI Behavior)

Attribute ID	NC_ATTR_RTSI_SIG_BEHAV
Hex Encoding	0x80000019
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute is to be used when a CAN object is used to output RTSI signals and defines if the RTSI line is to be pulsed or toggled. **Attribute values:** NC_RTSISIG_PULSE (Output RTSI Pulse): This pulses the RTSI line with a 100 µs pulse. NC_RTSISIG_TOGGLE (Toggle RTSI Line): This toggles the RTSI line. If the previous state was high, it will be toggled low, and vice versa.

NC_ATTR_RTSI_SIGNAL (RTSI Line Number)

Attribute ID	NC_ATTR_RTSI_SIGNAL
Hex Encoding	0x80000018
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute defines the RTSI B signal to be associated with the CAN object. **Attribute values:** 0 to 7 (for RTSI B signal lines). **Note:** In the hardware, four RTSI lines are for input and four lines are for output. Hence, four CAN objects can configure RTSI lines for input and four CAN objects can configure RTSI lines for output. An error will be reported when these limits are exceeded. **Note:** For low-speed and dual-speed boards, two lines are available as inputs and three (3) lines are available as outputs. The unavailable lines are used for low-speed transceiver fault reporting. **Note:** For PXI-CAN cards, RTSI signal 0 is unavailable. There is no limitation on which lines can be used as input or output.

NC_ATTR_RTSI_SKIP (RTSI Skip)

Attribute ID	NC_ATTR_RTSI_SKIP
Hex Encoding	0x80000021
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	This attribute defines the number of RTSI events to skip before logging them to the read queue for that object. Use this attribute with NC_ATTR_RTSI_MODE and with an attribute value of NC_RTSI_TIME_ON_IN. **Attribute values:** Any user number.

NC_ATTR_RX_CHANGES_ONLY (Receive Changes Only)

Attribute ID	NC_ATTR_RX_CHANGES_ONLY
Hex Encoding	80000015
Data Type	NCTYPE_BOOL
Permissions	Config, Get
Description	The NC_ATTR_RX_CHANGES_ONLY attribute applies only to CAN Object configurations in which the Communication Type (NC_ATTR_COMM_TYPE) is set to Receive CAN data frames. For those configurations, if NC_ATTR_RX_CHANGES_ONLY is set to NC_FALSE, NI-CAN places data from all incoming CAN data frames into the read queue. If this attribute is set to NC_TRUE, NI-CAN places data from an incoming CAN data frame into the read queue only if it differs from the previously received data. This attribute has no effect on the usage of a watchdog timeout for the CAN Object. For example, if this attribute is true and you also specify a watchdog timeout, NI-CAN restarts the watchdog timeout every time it receives a CAN data frame from the network, regardless of whether the data differs from the previous value.

NC_ATTR_STATE (Object State)

Attribute ID	NC_ATTR_STATE
Hex Encoding	80000009
Data Type	NCTYPE_STATE
Permissions	Get
Description	Current state of the CAN Object. In most cases, the NC_ST_STOPPED, NC_ST_WARNING, and NC_ST_ERROR states are merely reflected up from the underlying CAN Network Interface Object.

NC_ATTR_STATUS (Object Status)

Attribute ID	NC_ATTR_STATUS
Hex Encoding	8000000A
Data Type	NCTYPE_STATUS
Permissions	Get
Description	Background status of the CAN Object. Unless the NC_ST_WARNING or NC_ST_ERROR states are set in NC_ATTR_STATE, this attribute always returns CanSuccess. When you read an error or warning from this attribute, NI-CAN clears the appropriate state, and sets the background status back to CanSuccess. For communication errors such as CanErrComm, this background status is the same as the background status of the underlying CAN Network Interface Object. If a background error occurs, you can read it from this attribute, or obtain it from the next call to ncRead or ncWrite.

NC_ATTR_WRITE_PENDING (Write Entries Pending)

Attribute ID	NC_ATTR_WRITE_PENDING
Hex Encoding	80000012
Data Type	NCTYPE_UINT32
Permissions	Get
Description	Indicates the number of pending entries in the write queue. If NC_ST_WRITE_PENDING is zero, the NC_ST_WRITE_SUCCESS state is set (after NI-CAN successfully transmits the final frame).

NC_ATTR_WRITE_Q_LEN (Write Queue Length)

Attribute ID	NC_ATTR_WRITE_Q_LEN
Hex Encoding	80000014
Data Type	NCTYPE_UINT32
Permissions	Config, Get
Description	Length (maximum number of entries) for the write queue. For more information, refer to the description of the ncWrite function in Chapter 2, *NI-CAN Functions*.

Values for Communication Type

The following sections describe the allowable values for NC_ATTR_COMM_TYPE (Communication Type).

Receive Unsolicited (NC_CAN_COMM_RX_UNSOL)

Use this configuration to receive unsolicited CAN data frames from a remote device.

If the CAN data frames are expected periodically, you can use a watchdog timeout by setting Period (NC_ATTR_PERIOD) to the desired number of milliseconds. Then, when the CAN Object detects an incoming CAN data frame, it restarts the watchdog timeout. If the watchdog timeout expires before the next incoming CAN data frame is received for the CAN Object, NI-CAN reports a NC_ERR_TIMEOUT error. The watchdog timeout is used to verify that the remote node still exists and is transmitting data as expected. If you do not want to use a watchdog timeout, set Period to zero.

The Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute can be used to receive all data (NC_FALSE) or only changes (NC_TRUE).

Because this CAN Object does not transmit CAN data frames, the Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute is ignored (assumes NC_FALSE).

RTSI—This Object configuration supports two RTSI modes:

- NC_RTSI_TIME_ON_IN (On RTSI Input—Timestamp RTSI event)
- NC_RTSI_OUT_ON_RX (RTSI Output on Receiving CAN Frame)

Refer to *Attributes* in the *CAN Object* section of this chapter for information on using these RTSI attributes.

Receive Periodically Using Remote (NC_CAN_COMM_RX_PERIODIC)

Use this configuration to poll for data from a remote device periodically. Every period, the object transmits a CAN remote frame, and NI-CAN places the resulting CAN data frame response into the read queue.

The Period (NC_ATTR_PERIOD) attribute is used to configure the period between successive CAN remote frame transmissions.

The Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute can be used to receive all data (NC_FALSE), or only changes (NC_TRUE).

Because this CAN Object does not transmit CAN data frames, the Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute is ignored (assumes NC_FALSE).

Receive Value by Call Using Remote (NC_CAN_COMM_RX_BY_CALL)

Use this configuration to poll for data from a remote device using the ncWrite function. You must call ncWrite with DataSize zero to transmit a CAN remote frame. NI-CAN places the resulting CAN data frame response into the read queue.

If you want to specify the minimum amount of time between subsequent transmission of CAN remote frames, you can specify a minimum interval by setting Period (NC_ATTR_PERIOD) to the desired number of milliseconds. You configure the minimum interval as a promise to other nodes on the network that the object will not transmit its CAN frames with needless frequency, thus precluding transfer by lower priority CAN frames. You can use a write queue in conjunction with the minimum intervals to guarantee that the desired number of frames is transmitted on the network.

The Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) can be used to receive all data (NC_FALSE) or only changes (NC_TRUE).

Because this CAN Object does not transmit CAN data frames, the Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute is ignored (assumes NC_FALSE).

Transmit Data Periodically (NC_CAN_COMM_TX_PERIODIC)

Use this configuration to transmit a CAN data frame to a remote device periodically. The Period (NC_ATTR_PERIOD) attribute is used to configure the period between successive CAN data frame transmissions.

When NI-CAN transmits the last entry of the write queue, that entry is used every period until you provide a new entry using ncWrite. With this behavior, every entry is guaranteed to be transmitted at least once, and the object always has data available for transmission. If the write queue is empty when communication starts, the first periodic transmission does not occur until you provide a valid data value using ncWrite.

Because this CAN Object does not receive CAN data frames, the Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute is ignored (assumes NC_FALSE).

The Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute can be used to ignore incoming CAN remote frames (NC_FALSE), or to transmit previous data when a CAN remote frame is received (NC_TRUE).

RTSI—This Object configuration supports two RTSI modes:

- NC_RTSI_TX_ON_IN (On RTSI Input—Transmit CAN Frame)
- NC_RTSI_OUT_ON_TX (RTSI Output on Transmitting CAN Frame)

Refer to *Attributes* in the *CAN Object* section of this chapter for information on using these RTSI attributes.

Transmit Value by Response Only (NC_CAN_COMM_TX_RESP_ONLY)

Use this configuration to transmit CAN data frames only in response to an incoming CAN remote frame. When you call ncWrite, the data is placed in the write queue, and remains there until a CAN remote frame is received.

If the CAN remote frames are expected periodically, you can specify a watchdog timeout by setting Period (NC_ATTR_PERIOD) to the desired number of milliseconds. Then, when the CAN Object detects an incoming CAN remote frame, it restarts the watchdog timeout. If the watchdog timeout expires before the next incoming CAN remote frame is received for the CAN Object, NI-CAN reports an CanErrWatchdogTimeout error. The watchdog timeout is used to verify that the remote node still exists and is transmitting CAN remote frames as expected. If you do not want to use a watchdog timeout, set Period to zero.

Because this CAN Object does not receive CAN data frames, the Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute is ignored (assumes NC_FALSE).

Because this CAN Object always responds to incoming CAN remote frames, the Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute is ignored (assumes NC_TRUE).

Transmit Data by Call (NC_CAN_COMM_TX_BY_CALL)

Use this configuration to transmit a CAN data frame when ncWrite is called.

If you want to specify the minimum amount of time between subsequent transmission of CAN data frames, you can specify a minimum interval by setting Period (NC_ATTR_PERIOD) to the desired number of milliseconds (see the *Receive Value by Call Using Remote (NC_CAN_COMM_RX_BY_CALL)* section earlier in this chapter).

Because this CAN Object does not receive CAN data frames, the Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute is ignored (assumes NC_FALSE).

The Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute can be used to ignore incoming CAN remote frames (NC_FALSE), or to transmit previous data when a CAN remote frame is received (NC_TRUE).

RTSI—This Object configuration supports two RTSI modes:

- NC_RTSI_TX_ON_IN (On RTSI Input—Transmit CAN Frame)
- NC_RTSI_OUT_ON_TX (RTSI Output on Transmitting CAN Frame)

Refer to *Attributes* in the *CAN Object* section of this chapter for information on using these RTSI attributes.

Transmit Periodic Waveform (NC_CAN_COMM_TX_WAVEFORM)

Use this configuration to transmit a fixed sequence of CAN data frames over and over, one CAN data frame every period. By varying the data value in each CAN data frame, this configuration can be used to transmit a waveform to a remote device.

The Period (NC_ATTR_PERIOD) attribute is used to configure the period between successive CAN data frame transmissions.

Because this CAN Object does not receive CAN data frames, the Receive Changes Only (NC_ATTR_RX_CHANGES_ONLY) attribute is ignored (assumes NC_FALSE).

The Transmit by Response (NC_ATTR_CAN_TX_RESPONSE) attribute can be used to ignore incoming CAN remote frames (NC_FALSE), or to transmit previous data when a CAN remote frame is received (NC_TRUE).

The following steps illustrate the typical usage of Transmit Periodic Waveform.

1. Configure the CAN Network Interface Object with Start On Open false, then configure the object.
2. Configure the CAN Object as Transmit Periodic Waveform and set a nonzero Write Queue length, then open the Object.
3. Call ncWrite for the CAN Object, once for every entry specified for the Write Queue Length.
4. Use ncAction to start the CAN Network Interface Object (not the CAN Object).

   The CAN Object transmits the first entry in the write queue, then waits the specified Period, then transmits the second entry, and so on. After the last entry is transmitted, the CAN Object waits the specified Period, then transmits the first entry again.
5. You can use ncAction to stop and restart the CAN Object's transmissions. When the CAN Object is stopped, you can use ncWrite to provide new waveform entries. When the write queue is full, ncWrite always replaces the first (oldest) entry in the queue.

RTSI—This Object configuration supports two RTSI modes:

- `NC_RTSI_TX_ON_IN` (On RTSI Input—Transmit CAN Frame)
- `NC_RTSI_OUT_ON_TX` (RTSI Output on Transmitting CAN Frame)

Refer to *Attributes* in the *CAN Object* section of this chapter for information on using these RTSI attributes.

Examples of Different Communication Types

The following figures demonstrate how you can use the Communication Type attribute for actual network data transfer. Each figure shows two separate NI-CAN applications that are physically connected across a CAN network.

Figure 3-1 shows a CAN Object that periodically transmits data to another CAN Object. The receiving CAN Object can queue up to five data values.

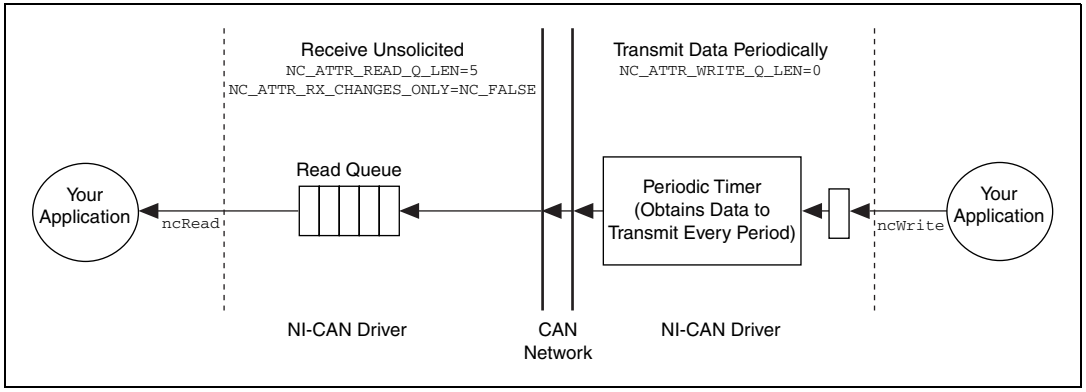


Figure 3-1. Example of Periodic Transmission

Figure 3-2 shows a CAN Object that polls data from another CAN Object. NI-CAN transmits the CAN remote frame when you call `ncWrite`.

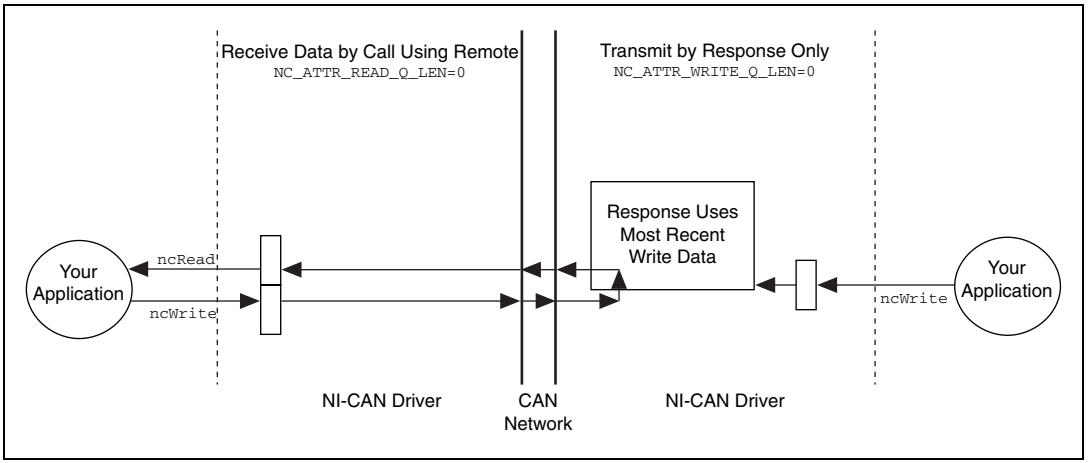


Figure 3-2. Example of Polling Remote Data Using ncWrite

Figure 3-3 shows a CAN Object that polls data from another CAN Object. NI-CAN transmits the remote frame periodically and places only changed data into the read queue.

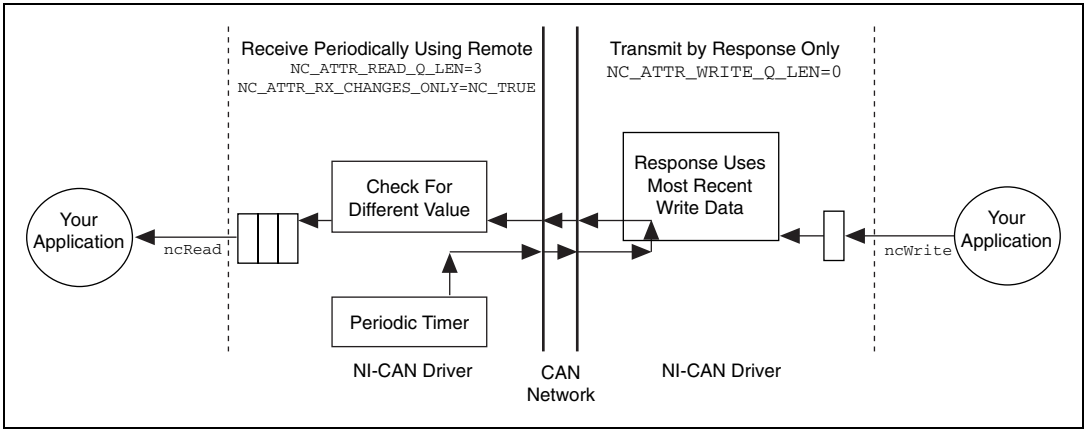


Figure 3-3. Example of Periodic Polling of Remote Data

4

RTSI Programming

This chapter consolidates the Real Time System Integration (RTSI) programming features available in NI-CAN. This information is also in the *CAN Network Interface Object* and *CAN Object* attributes sections of Chapter 3, *NI-CAN Objects*, and is useful when used in conjunction with the object specifics.

Description

RTSI is a bus that interconnects National Instruments DAQ, IMAQ, Motion, and CAN boards. This feature allows synchronization of DAQ, IMAQ, Motion, and CAN boards by allowing exchange of timing signals. Using RTSI, a device (board) can control one or more slave devices. PCI/AT boards require a ribbon cable for the connections, but for PXI boards the connections are available on the PXI chassis backplane. Refer to the *NI-CAN User Manual* for your version of the Windows operating system for more details on the hardware connector.

In NI-CAN, RTSI is configured via attributes in either the Network Interface or the CAN Object configuration.

In C, the configuration is done by adding new elements to `AttributeIdList` and `AttrValueList` before calling `ncConfig`.

In LabVIEW, the configuration is done via the Network Interface Config cluster or the CAN Object Configuration Cluster, which has an optional input to wire an RTSI configuration cluster. Refer to the `ncConfig` function for further details.

The following is a summary of the attributes and possible values that these attributes can have. For hex encoding of the attributes, refer to the *CAN Network Interface Object* or *CAN Object* attributes sections of Chapter 3, *NI-CAN Objects*.

Attributes

NC_ATTR_RTSI_MODE (RTSI Mode)

This attribute defines whether a CAN object is to be configured as a RTSI slave or master. The following values can be used:

Attribute Values in C	Attribute Values in LabVIEW
NC_RTSI_NONE	Disable RTSI
NC_RTSI_TX_ON_IN	On RTSI Input—Transmit CAN Frame
NC_RTSI_TIME_ON_IN	On RTSI Input—Timestamp RTSI event
NC_RTSI_OUT_ON_RX	RTSI Output on Receiving CAN Frame
NC_RTSI_OUT_ON_TX	RTSI Output on Transmitting CAN Frame
NC_RTSI_OUT_ACTION_ONLY	RTSI Output on ncAction call

The attribute values are described below.

NC_RTSI_NONE (Disable RTSI)

No RTSI use is needed for this object. If the object is set to this value, all other RTSI configuration is ignored.

NC_RTSI_TX_ON_IN (On RTSI Input—Transmit CAN Frame)

Network Interface: In this mode, NI-CAN will transmit the most recent frame on an incoming RTSI trigger on the RTSI line configured via NC_ATTR_RTSI_SIGNAL. To begin transmission, a frame must be written to the write queue of the object (by calling ncWrite) and an RTSI signal applied to the configured RTSI line. NI-CAN will retransmit the last frame until a new frame is enqueued.

CAN Object: In this mode, the CAN Object can be configured as:

- NC_CAN_COMM_TX_BY_CALL (Transmit Frame by calling ncWrite)
- NC_CAN_COMM_TX_PERIODIC (Periodic Transmission)
- NC_CAN_COMM_TX_WAVEFORM (Waveform Transmit)

In all of the three configurations, a CAN frame is transmitted on every incoming RTSI trigger. The period is ignored (if nonzero). Refer to the *CAN Object* section of Chapter 3, *NI-CAN Objects*, for instructions to set up the CAN object for each communication type.

NC_RTSI_TIME_ON_IN (On RTSI Input—Timestamp RTSI Event)

Network Interface: In this mode, NI-CAN will timestamp an incoming RTSI trigger on the RTSI line configured via `NC_ATTR_RTSI_SIGNAL` and enqueue (in the object's read queue) a frame containing special entries in the following fields (as noted):

Timestamp:	Time when RTSI event occurred
ArbitrationId:	`0x40000001`
FrameType:	3 `(NC_FRMTYPE_RTSI)`
DataLength:	RTSI line number that produces the event
Data[8]:	unchanged

CAN Object: Because the CAN object's receiving structure contains only the Timestamp and Data[8] fields, you must specify a 4-byte data frame that NI-CAN can use in the first four bytes of the Data[8] field, to help distinguish the RTSI event from other data frames. This user frame is configured via the `NC_ATTR_RTSI_FRAME` attribute in the CAN object RTSI configuration.

This attribute is ignored when a Network Interface is being configured.

NC_RTSI_OUT_ON_RX (RTSI Output on Receiving CAN Frame)

Network Interface & CAN Objects: In this mode, NI-CAN will output an RTSI trigger on the line configured via `NC_ATTR_RTSI_SIGNAL` whenever a frame is enqueued in the object read queue.

NC_RTSI_OUT_ON_TX (RTSI Output on Transmitting CAN Frame)

Network Interface & CAN Objects: In this mode, NI-CAN will output an RTSI trigger on the line configured via `NC_ATTR_RTSI_SIGNAL` whenever a frame is successfully transmitted.

NC_RTSI_OUT_ACTION_ONLY (RTSI Output on ncAction call)

Network Interface & CAN Objects: In this mode, NI-CAN will output an RTSI trigger on the line configured via `NC_ATTR_RTSI_SIGNAL` whenever user calls the `ncAction` function. With this function, a user can set/toggle a RTSI line high or low.

NC_ATTR_RTSI_SIGNAL (RTSI Line Number)

This attribute defines the RTSI signal that must be associated with the CAN object. This attribute must be used with the Network Interface and CAN objects for all desired RTSI modes.

Attribute values: 0 to 7 (for RTSI signal lines).

Note In the hardware, four RTSI lines are for input and four lines are for output. Hence, four CAN objects can configure RTSI lines for input and four CAN objects can configure RTSI lines for output. An error will be reported when these limits are exceeded.

Note For low-speed and dual-speed boards, three (3) lines are available as inputs and two (2) lines are available as outputs. The unavailable lines are used for low-speed transceiver fault reporting.

Note For PXI-CAN cards, RTSI signal `0` is unavailable.

There is no limitation on which RTSI line number can be used as input or output.

NC_ATTR_RTSI_SIG_BEHAV (RTSI Behavior)

Use this attribute when a CAN object is used to output RTSI signals when CAN is the RTSI driver. This attribute must be used with the Network Interface and CAN objects that output an RTSI trigger.

Attribute Values in C	Attribute Values in LabVIEW
`NC_RTSISIG_PULSE`	Output RTSI Pulse
`NC_RTSISIG_TOGGLE`	Toggle RTSI Line

The attribute values are:

- NC_RTSISIG_PULSE (Output RTSI Pulse): This pulses the RTSI line with a 100 µs pulse.
- NC_RTSISIG_TOGGLE (Toggle RTSI Line): This toggles the RTSI line. If the previous state was high, it will be toggled low, and vice versa.

NC_ATTR_RTSI_FRAME (UserRTSIFrame)

Use this attribute when a CAN object is to be configured with the attribute NC_ATTR_RTSI_MODE and with an attribute value of NC_RTSI_TIME_ON_IN.

Because the CAN object's receiving structure contains only the Timestamp and Data[8] fields, you must specify a 4-byte data frame that NI-CAN can use in the first four bytes of the Data[8] field, to help distinguish the RTSI event from other data frames. This user frame is configured via the NC_ATTR_RTSI_FRAME attribute in the RTSI configuration for the CAN object.

Attribute values: Any user-defined unsigned32 number in hex. For example, 0xAABBCCDD.

NC_ATTR_RTSI_SKIP (RTSI Skip)

This attribute defines the number of RTSI events to skip before logging them to the read queue for that object. Use this attribute with NC_ATTR_RTSI_MODE and an attribute value of NC_RTSI_TIME_ON_IN.

Attribute values: Any user number.

Examples

The following RTSI examples are available:

- C programming (in the \\nican\examples\CAN DAQ Synchronization folder)
- The readme.txt file in the examples folder describes each example, including the names of the associated files. You can build the examples using LabWindows/CVI, Microsoft Visual C++, or Borland C++.
- In LabVIEW 6.0, you can access the RTSI examples at **Help» Examples»Other NI Products»Controller Area Network (CAN)**. Other versions of LabVIEW include similar help.

For information about using the examples and which tester to use on the DAQ board to test the functionality, refer to documentation in the C source file and VI-Info (for LabVIEW examples).

A

NI-CAN Object States

This appendix describes the NI-CAN object states.

Every object in NI-CAN contains a state attribute (NC_ATTR_STATE) with the following format. The bits marked as 0 are reserved for future use.

31–6	5	4	3	2	1	0
0	WARNING	ERROR	READ MULT	STOPPED	WRITE SUCCESS	READ AVAIL

Figure A-1. State Format

You can detect the object states using one of the following schemes:

- Call ncGetAttribute to get the NC_ATTR_STATE attribute.
- Call ncWaitForState to wait for one or more states to occur.
- For C, use ncCreateNotification to register a callback for one or more states.
- For LabVIEW, use ncCreateOccurrence to create a LabVIEW occurrence for one or more states.

Table A-1 describes each object state.

Table A-1. NI-CAN Object States

Constant	Bitmask	Description
NC_ST_READ_AVAIL	0x00000001 (Bit 0)	Indicates that new data is available to be read using ncRead. Set when data is received from network, and cleared when all available data is read.
NC_ST_WRITE_SUCCESS	0x00000002 (Bit 1)	Indicates that all data provided using ncWrite has been successfully transmitted onto network. Set when last transmission is successful, and cleared by any call to ncWrite.

Table A-1. NI-CAN Object States (Continued)

Constant	Bitmask	Description
NC_ST_STOPPED	0x00000004 (Bit 2)	Indicates that object is in stopped state (not communicating on network). This state can occur as result of calling ncAction with NC_OP_STOP, or due to serious communication error, such as CAN bus off, which causes object to stop. If this state is clear, the object is in its normal running state.
NC_ST_READ_MULT	0x00000008 (Bit 3)	Indicates that a specified number of frames have arrived in the read queue. The threshold number is specified by setting the NC_ATTR_READ_MULT_SIZE attribute. This state is used when you plan to call the ncreadMult function to read frames. This state is cleared after ncReadMult if less than NC_ATTR_READ_MULT_SIZE frames remain in the read queue.
NC_ST_ERROR	0x00000010 (Bit 4)	Indicates that an error status has occurred in background. Set when error occurs, and cleared when you obtain status value. Status value is obtained by getting NC_ATTR_STATUS attribute, or on next call to ncRead or ncWrite. This state indicates background problems such as communication errors, and is not set for problems that are associated with individual function calls (such as an invalid parameter).
NC_ST_WARNING	0x00000020 (Bit 5)	Indicates that warning status has occurred in background. Set when warning occurs, and cleared when you obtain status value. Status value is obtained by getting NC_ATTR_STATUS attribute, or on next call to ncRead or ncWrite. This state indicates background problems such as communication warnings, and is not set for problems that are associated with individual function calls (such as an invalid parameter).

B

NI-CAN Status

This appendix describes the NI-CAN status codes and the qualifiers for each code.

Each NI-CAN function returns a value that indicates the status of the function call. Your application should check this status after each NI-CAN function call. The following sections describe the NI-CAN status, and how you can check it in your application.

Note The NI-CAN status format changed from version 1.4 to version 1.5. If you are developing a new NI-CAN application, this change will not affect your development. If you have an existing NI-CAN application that was developed prior to July 2001, you can either 1.) run a utility to enable backward compatibility for NI-CAN status, or 2.) adapt your code to the current NI-CAN status. For more information, refer to the `errors.txt` in the NI-CAN Installation directory.

Checking Status in LabVIEW

For applications written in LabVIEW, status checking is basically handled automatically. For all NI-CAN functions, the lower left and right terminals provide status information using LabVIEW Error Clusters. LabVIEW Error Clusters are designed so that status information flows from one function to the next, and function execution stops when an error occurs. For more information, refer to the *Error Handling* section in the LabVIEW online help.

The LabVIEW Error Clusters returned by NI-CAN functions use the same format as other National Instruments products. You can wire `Error out` from any NI-CAN function to the standard LabVIEW error functions, such as `Simple Error Handler`. The message returned by `Simple Error Handler` will describe the error, including possible solutions.

Table B-1 summarizes NI-CAN's use of each Error Cluster parameter.

Table B-1. NI-CAN Error Cluster

Code	Status	Source	Meaning
Negative	True	Name of function where error occurred	Error. Function did not perform expected behavior.
Positive	False	Name of function where warning occurred	Warning. Function performed as expected, but a condition arose that may require your attention.
Zero	False	Empty string	Success. Function completed successfully.

Within your LabVIEW Block Diagram, wire the `Error in` and `Error out` terminals of all NI-CAN functions together in succession. When an error is detected in any NI-CAN function (`status` parameter true), all subsequent NI-CAN functions are skipped except for `ncClose`. The `ncClose` function executes regardless of whether the incoming `status` is true or false. This ensures that all NI-CAN objects are closed properly when execution stops due to an error.

When a warning occurs in an NI-CAN function, execution proceeds normally. To detect suspected warnings in your application, you can write code in your block diagram to examine the `code` parameter, or you can use the `Probe Data` tool on an `Error out` terminal during execution.

Checking Status in C or C++

For applications written in C or C++, each NI-CAN function returns a status code as a signed 32-bit integer. Table B-2 summarizes the NI-CAN use of this status:

Table B-2. NI-CAN Status Codes

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

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

To assist with debugging, NI-CAN provides a function you can use to display a message that describes the error/warning, including possible solutions. This `ncStatusToString` function takes a status code as input, then returns a descriptive string. For more information on `ncStatusToString`, refer to the *NI-CAN Programmer Reference Manual*.

The following piece of code shows an example of handling NI-CAN status during application debugging:

```
status= ncOpenObject ("CAN0", &MyObjHandle);
PrintStat (status, "ncOpen CAN0");
```

where the function `PrintStat` has been defined at the top of the program as:

```
void PrintStat(NCTYPE_STATUS status, char *source)
{
    char statusString[512];
    if(status !=0)
    {
        ncStatusToString(status, sizeof(statusString),
                         StatusString);
        printf("\n%s\nSource = %s\n", statusString,
                source);
        if (status < 0)
        {
            ncCloseObject(MyObjHandle);
            exit(1);
        }
    }
}
```

In some situations, you may want to check for specific errors in your code. For example, when `ncWaitForState` times out, you may want to continue communication, rather than exit the application. To check for specific errors, use the constants defined in `nican.h`. These constants have the same names as described in the *NI-CAN Programmer Reference Manual*. For example, to check for a function timeout:

```
if (status == CanErrFunctionTimeout)
```

C

Technical Support Resources

Web Support

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

NI Developer Zone

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

Customer Education

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

System Integration

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

Worldwide Support

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

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

Glossary

Prefix	Meanings	Value
n-	nano-	10^{-9}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

A

action — *See* method.

actuator — A device that uses electrical, mechanical, or other signals to change the value of an external, real-world variable. In the context of device networks, actuators are devices that receive their primary data value from over the network; examples include valves and motor starters. Also known as *final control element*.

Application Programming Interface (API) — A collection of functions used by a user application to access hardware. Within NI-CAN, you use API functions to make calls into the NI-CAN driver.

arbitration ID — An 11- or 29-bit ID transmitted as the first field of a CAN frame. The arbitration ID determines the priority of the frame, and is normally used to identify the data transmitted in the frame.

attribute — The externally visible qualities of an object; for example, an instance Mary of class Human could have the attributes Gender and Age, with the values Female and 31. Also known as *property*.

B

b	Bits.
bus off	A CAN node goes into the bus off state when its transmit error counter increments above 255. The node does not participate in network traffic, because it assumes that a defect exists that must be corrected.

C

CAN	Controller Area Network.
CAN/LS	*See* Low-speed CAN.
CAN data frame	Frame used to transmit the actual data of a CAN Object. The RTR bit is clear, and the data length indicates the number of data bytes in the frame.
CAN frame	In addition to fields used for error detection/correction, a CAN frame consists of an arbitration ID, the RTR bit, a four-bit data length, and zero to eight bytes of data.
CAN Network Interface Object	Within NI-CAN, an object that encapsulates a CAN network interface on the host computer.
CAN Object	A CAN identifier, along with its associated data.
CAN remote frame	Frame used to request data for a CAN Object from a remote node; the RTR bit is set, and the data length indicates the amount of data desired (but no data bytes are included).
class	A set of objects that share a common structure and a common behavior.
connection	An association between two or more nodes on a network that describes when and how data is transferred.
controller	A device that receives data from sensors and sends data to actuators in order to hold one or more external, real-world variables at a certain level or condition. A thermostat is a simple example of a controller.

D

device	*See* node.
device network	Multi-drop digital communication network for sensors, actuators, and controllers.
DLL	Dynamic link library.
DMA	Direct memory access.

E

error active	A CAN node is in error active state when both the receive and transmit error counters are below 128.
error counters	Every CAN node keeps a count of how many receive and transmit errors have occurred. The rules for how these counters are incremented and decremented are defined by the CAN protocol specification.
error passive	A CAN node is in error passive state when one or both of its error counters increment above 127. This state is a warning that a communication problem exists, but the node is still participating in network traffic.
extended arbitration ID	A 29-bit arbitration ID. Frames that use extended IDs are often referred to as CAN 2.0 Part B (the specification that defines them).

F

FCC	Federal Communications Commission.
frame	A unit of information transferred across a network from one node to another; the protocol defines the meaning of the bit fields within a frame. Also known as *packet*.

H

hex	Hexadecimal.
Hz	Hertz.

I

instance	An abstraction of a specific real-world thing; for example, John is an instance of the class Human. Also known as *object*.
ISO	International Standards Organization.

K

KB	Kilobytes of memory.

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench.
local	Within NI-CAN, anything that exists on the same host (personal computer) as the NI-CAN driver.
Low-speed CAN	Implementation of CAN as defined in ISO 11519.

M

MB	Megabytes of memory.
method	An action performed on an instance to affect its behavior; the externally visible code of an object. Within NI-CAN, you use NI-CAN functions to execute methods for objects. Also known as *service*, *operation*, and *action*.
minimum interval	For a given connection, the minimum amount of time between subsequent attempts to transmit frames on the connection. Some protocols use minimum intervals to guarantee a certain level of overall network performance.
multi-drop	A physical connection in which multiple devices communicate with one another along a single cable.

N

network interface	A node's physical connection onto a network.
NI-CAN driver	Device driver and/or firmware that implement all the specifics of a CAN network interface. Within NI-CAN, this software implements the CAN Network Interface Object as well as all objects above it in the object hierarchy.
node	A physical assembly, linked to a communication line (cable), capable of communicating across the network according to a protocol specification. Also known as *device*.
notification	Within NI-CAN, an operating system mechanism that the NI-CAN driver uses to communicate events to your application. You can think of a notification of as an API function, but in the opposite direction.

O

object	*See* instance.
object-oriented	A software design methodology in which classes, instances, attributes, and methods are used to hide all of the details of a software entity that do not contribute to its essential characteristics.

P

peer-to-peer	Network connection in which data is transmitted from the source to its destination(s) without need for an explicit request. Although data transfer is generally unidirectional, the protocol often uses low level acknowledgments and error detection to ensure successful delivery.
periodic	Connections that transfer data on the network at a specific rate.
polled	Request/response connection in which a request for data is sent to a device, and the device sends back a response with the desired value.
protocol	A formal set of conventions or rules for the exchange of information among nodes of a given network.

R

RAM	Random-access memory.
remote	Within NI-CAN, anything that exists in another node of the device network (not on the same host as the NI-CAN driver).
Remote Transmission Request (RTR) bit	This bit follows the arbitration ID in a frame, and indicates whether the frame is the actual data of the CAN Object (CAN data frame), or whether the frame is a request for the data (CAN remote frame).
request/response	Network connection in which a request is transmitted to one or more destination nodes, and those nodes send a response back to the requesting node. In industrial applications, the responding (slave) device is usually a sensor or actuator, and the requesting (master) device is usually a controller. Also known as *master/slave*.
resource	Hardware settings used by National Instruments CAN hardware, including an interrupt request level (IRQ) and an 8 KB physical memory range (such as D0000 to D1FFF hex).

S

s	Seconds.
sensor	A device that measures electrical, mechanical, or other signals from an external, real-world variable; in the context of device networks, sensors are devices that send their primary data value onto the network; examples include temperature sensors and presence sensors. Also known as *transmitter.*
standard arbitration ID	An 11-bit arbitration ID. Frames that use standard IDs are often referred to as CAN 2.0 Part A; standard IDs are by far the most commonly used.

U

unsolicited	Connections that transmit data on the network sporadically based on an external event. Also known as *nonperiodic*, *sporadic*, and *event driven*.

V

VI	Virtual Instrument.

W

watchdog timeout	A timeout associated with a connection that expects to receive network data at a specific rate. If data is not received before the watchdog timeout expires, the connection is normally stopped. You can use watchdog timeouts to verify that the remote node is still operational.

Index

A

B

C

D

E

F

G

H

L

M

N

P

Q

R

S

T

U

W