CAN

NI-CAN™ Programmer Reference Manual for Win32

November 1997 Edition
Part Number 321369B-01

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

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

Fax-on-Demand Support
512 418 1111

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

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

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

LabVIEW™ and NI-CAN™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Table of Contents

Appendix B
Status Codes and Qualifiers

Appendix C
Customer Communication

Glossary

Index

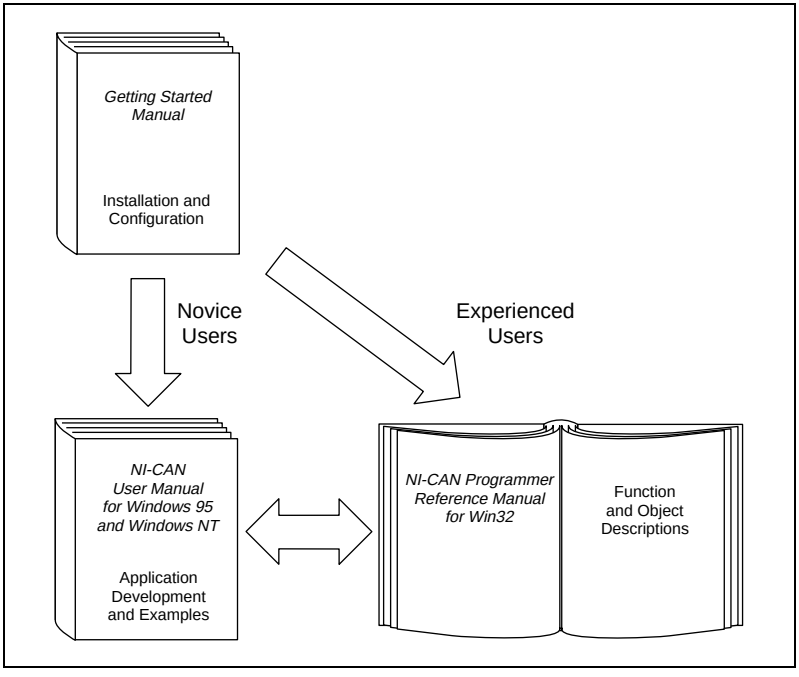
Figures

Tables

About This Manual

This manual is a programming reference for functions, objects, and data types in the NI-CAN software for Win32, the 32-bit programming environment of Windows 95 and Windows NT. The NI-CAN software for Windows 95 is meant to be used with Windows 95. The NI-CAN software for Windows NT is meant to be used with Windows NT version 3.51 or higher. This manual assumes that you are already familiar with the Windows system you are using.

How to Use the Manual Set



Use the getting started manual to install and configure your CAN hardware and NI-CAN software.

Use the *NI-CAN User Manual for Windows 95 and Windows NT* 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 Win32* for specific information about each NI-CAN function and object, such as format, parameters, and possible errors.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *NI-CAN Host Data Types*, describes the host data types used by NI-CAN functions and objects.
- Chapter 2, *NI-CAN Functions*, lists the NI-CAN functions and describes the format, purpose, parameters, and return status for each function.
- Chapter 3, *NI-CAN Objects*, 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.
- Appendix A, *NI-CAN Object States*, describes the NI-CAN object states.
- Appendix B, *Status Codes and Qualifiers*, describes the NI-CAN status codes and the qualifiers for each code.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual.

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options» Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows 95 tabs, or LEDs.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text for which you supply the appropriate word or value, such as in Windows 3.*x*.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions, and for statements and comments taken from program code.

The *Glossary* lists abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms.

Related Documentation

The following documents contain information that you may 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*
- *CAN Specification Version 2.0*, 1991, Robert Bosch Gmbh., Postfach 500, D-7000 Stuttgart 1
- LabVIEW Online Reference
- Win32 Software Development Kit (SDK) online help

Customer Communication

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

Chapter 1

NI-CAN Host Data Types

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

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

Table 1-1. NI-CAN Host 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.

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

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
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.
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. Letters are encoded as numeric equivalents (‘A’ is 1, ‘Z’ is 26, etc.). Version 2.0B would be hexadecimal 02000200, 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*.

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

NI-CAN Data Type	ANSI C Binding	LabVIEW Binding	Description
NCTYPE_ATTRID	NCTYPE_UINT32	U32	Attribute identifier.
NCTYPE_OPCODE	NCTYPE_UINT32	U32	Operation code used with ncAction function.
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, *Status Codes and Qualifiers.*
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 Host 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_FRAME_TIMED	struct	Output terminals of ncReadNet.vi	Structure used with ncRead and 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	Structure used with ncRead and CAN Object For information, refer to description of CAN Object in Chapter 3, *NI-CAN Objects.*

Chapter 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.

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.

Return Status

After every NI-CAN function description, all possible return status codes are listed. For complete information on status format and the qualifiers used with each status code, refer to Appendix B, *Status Codes and Qualifiers*.

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. The example programs are described in Chapter 4, *Application Examples*, in the *NI-CAN User Manual for Windows 95 and Windows NT*.

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 prior to its use.
`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's attribute.
`ncOpenObject`	Open an object.
`ncRead`	Read the data value of an object.
`ncSetAttribute`	Set the value of an object's attribute.
`ncWaitForState`	Wait for one or more states to occur in an object.
`ncWrite`	Write the data value of an object.

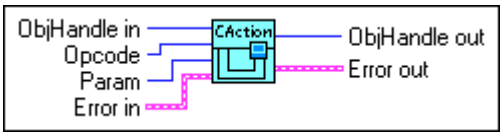
ncAction

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.

ncAction

Continued

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 in order 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.

ncAction

Continued

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. In order 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` has been 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.

Return Status

`NC_SUCCESS`	Success (no warning or error).
`NC_ERR_BAD_PARAM`	Invalid parameter.
`NC_ERR_BAD_VALUE`	Invalid values for configuration attributes. Returned only when `Opcode` is `NC_OP_START`.
`NC_ERR_DRIVER`	Implementation-specific error in the NI-CAN driver.

ncAction

Continued

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);
```

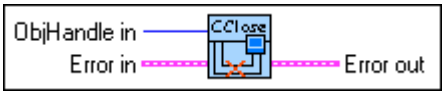
ncCloseObject

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 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.

Return Status

`NC_SUCCESS`	Success (no warning or error).
`NC_ERR_BAD_PARAM`	Invalid parameter.
`NC_ERR_DRIVER`	Implementation-specific error in the NI-CAN driver.

ncCloseObject

Continued

Example

This example assumes the following declarations:

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

Close an object.

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

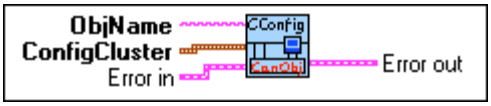
ncConfig

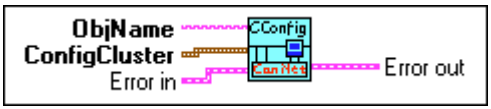
Purpose

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).

ncConfig

Continued

Description

ncConfig initializes the configuration attributes of an object before opening it. If you have configured objects using the NI-CAN Configuration utility, you do not need to call this function in your application. You can use the ncConfig function in applications that must be entirely self-contained, and thus cannot use the external NI-CAN Configuration utility. For any object, ncConfig overrides the configuration specified in the NI-CAN Configuration utility, if any.

ObjName uses the same object hierarchy syntax as ncOpenObject; it cannot be a user-defined alias.

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) and one for the CAN Object (ncObjAttr.ctl).

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.

ncConfig

Continued

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)

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)

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

Return Status

`NC_SUCCESS`	Success (no warning or error).
`NC_ERR_BAD_NAME`	Invalid or unrecognized name in `ObjName`.
`NC_ERR_BAD_PARAM`	Invalid parameter.
`NC_ERR_ALREADY_OPEN`	Object already opened.
`NC_ERR_DRIVER`	Implementation-specific error in the NI-CAN driver.
`NC_ERR_BAD_VALUE`	Invalid values for configuration attributes.

ncConfig

Continued

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);
```

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.
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.

ncCreateNotification

Continued

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.

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 in order 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.
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.

ncCreateNotification

Continued

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 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 (NC_ERR_TIMEOUT status code with NC_QUAL_TIMO_FUNCTION qualifier). If no background error or warning is reported, and no notification error occurred, Status is NC_SUCCESS.

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

ncCreateNotification

Continued

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.

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.

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

Return Status

NC_SUCCESS	Success (no warning or error).
NC_ERR_BAD_PARAM	Invalid parameter.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.

ncCreateNotification

Continued

Example

Create a notification for the NC_ST_READ_AVAIL state.

```
NCTYPE_STATE    _NCFUNC_ MyCallback (NCTYPE_OBJHObjHandle,
                                     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);
.
.
.
}
```

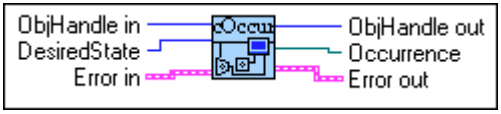
ncCreateOccurrence

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.

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`.

ncCreateOccurrence

Continued

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.

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, if DesiredState is NC_ST_READ_AVAIL, you should call ncRead 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.

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.

ncCreateOccurrence

Continued

NC_ST_WARNING	Warning occurred in background.

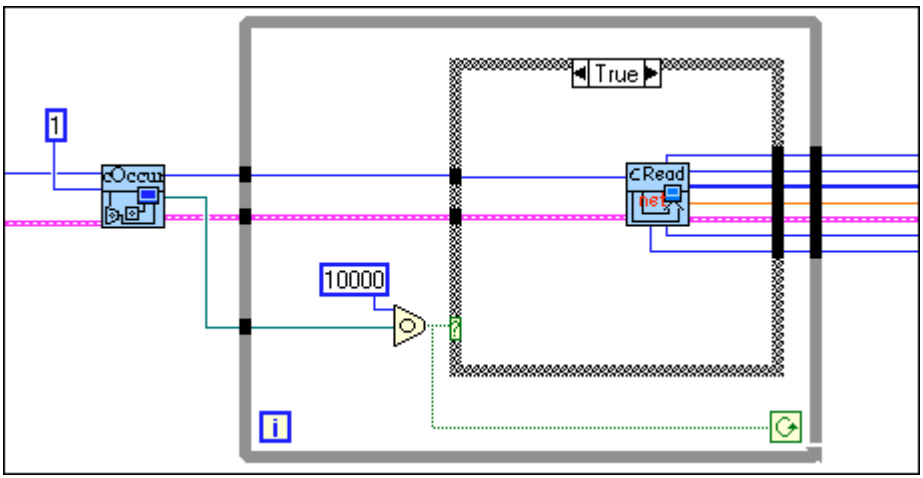
For more information on these states and the bit values used for each, refer to Appendix A, *NI-CAN Object States*.

Return Status

NC_SUCCESS	Success (no warning or error).
NC_ERR_BAD_PARAM	Invalid parameter.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.

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.



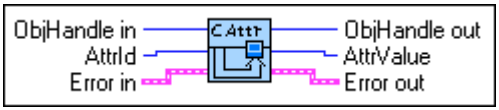
ncGetAttribute

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.

ncGetAttribute

Continued

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*.

Return Status

`NC_SUCCESS`	Success (no warning or error).
`NC_ERR_BAD_PARAM`	Invalid parameter.
`NC_ERR_DRIVER`	Implementation-specific error in the NI-CAN driver.

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);
```

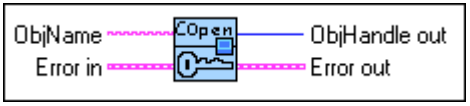
ncOpenObject

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.

You can use two syntax schemes can used for `ObjName`: the object hierarchy syntax and the user-defined alias syntax.

Use the object hierarchy syntax to open any object supported by NI-CAN. The object hierarchy 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.

ncOpenObject

Continued

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*.

The second scheme you can use for `ObjName` is that of user-defined aliases. You create a user-defined alias with the NI-CAN Configuration utility for use as an alias to a complete object hierarchy.

The syntax for user-defined aliases consists of a single ASCII name preceded by '#'. The '#' character differentiates user-defined aliases from the predefined names of the object hierarchy.

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 `NC_ERR_ALREADY_OPEN`. 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*.

Return Status

`NC_SUCCESS`	Success (no warning or error).
`NC_ERR_BAD_NAME`	Invalid or unrecognized name in `ObjName`.
`NC_ERR_BAD_PARAM`	Invalid parameter.

ncOpenObject

Continued

NC_ERR_ALREADY_OPEN	Object already opened by another application.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.

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. Open an alias to the CAN Object at standard arbitration ID 14 on CAN1. This alias was specified within the NI-CAN Configuration utility.

   ```
   status = ncOpenObject ("#EngineSpeed", &objh);
   ```

5. This call returns an error of NC_ERR_BAD_NAME with qualifier 2 (80020003 hex), because the Z makes the CAN Object name invalid.

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

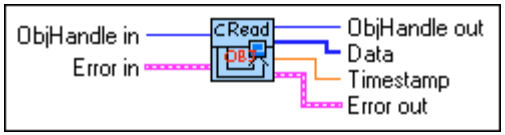
ncRead

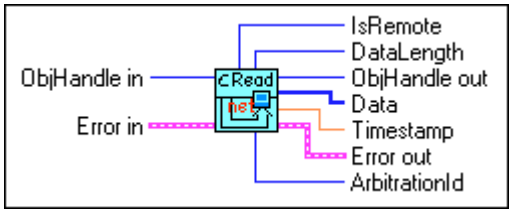
Purpose

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.

ncRead

Continued

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.

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 NC_ERR_OLD_DATA 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 NC_ERR_OVERFLOW error, along with the qualifier NC_QUAL_OVFL_READ. 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 NC_ERR_OLD_DATA 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

ncRead

Continued

(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). For more information, refer to the LabVIEW Online Reference.

CAN Network Interface Object

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

ncRead

Continued

Table 2-4. NCTYPE_CAN_FRAME_TIMED 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 Host Data Types*.
IsRemote	NCTYPE_BOOL	Indicates whether frame is CAN remote frame (NC_TRUE) or CAN data frame (NC_FALSE). It is always false for ncRead, indicating a CAN data frame. The CAN Network Interface Object cannot receive incoming CAN remote frames.
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.

ncRead

Continued

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.

Return Status

NC_SUCCESS	Success (no warning or error).
NC_ERR_BAD_PARAM	Invalid parameter.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.
NC_ERR_OLD_DATA	Data returned from ncRead is the same as the data returned from the previous call to ncRead.
NC_ERR_OVERFLOW	Read queue overflow. This error code does not apply to ncRead itself, but indicates an error in background communication. A valid data value is still returned to you from ncRead, and all other data received prior to the overflow remains in the read queue.
NC_ERR_TIMEOUT	Watchdog timeout expired for a CAN Object. This error code does not apply to ncRead itself, but indicates an error in background communication.
NC_ERR_CAN_BUS_OFF	Error or warning indicating CAN communication errors. This error code does not apply to ncRead itself, but indicates an error in background communication.

Examples

These examples assume the following declarations:

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

ncRead

Continued

```
NCTYPE_CAN_FRAME_TIMED     rframe;
NCTYPE_CAN_DATA_TIMED      rdata;
```

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);
   ```

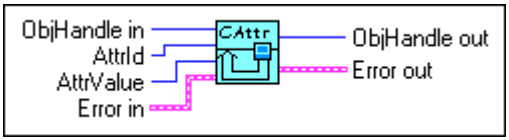
ncSetAttribute

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.

ncSetAttribute

Continued

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*.

Return Status

NC_SUCCESS	Success (no error or warning).
NC_ERR_BAD_PARAM	Invalid parameter. This error is returned when the attribute specified by AttrId has Get or Config permissions.
NC_ERR_BAD_VALUE	The value of the attribute is invalid for the specified AttrId.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.

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);
```

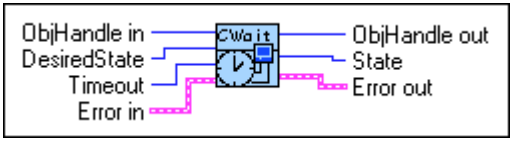
ncWaitForState

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 (bitmask).
`Timeout`	Length of time to wait.

Output

`StatePtr (State)`	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 `State`.

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.

ncWaitForState

Continued

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.

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) and `ncCreateOccurrence` (LabVIEW only) functions.

Return Status

`NC_SUCCESS`	Success (no error or warning).
`NC_ERR_BAD_PARAM`	Invalid parameter.
`NC_ERR_TIMEOUT`	`Timeout` expired before any desired states occurred.
`NC_ERR_DRIVER`	Implementation-specific error in the NI-CAN driver.

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);
   ```

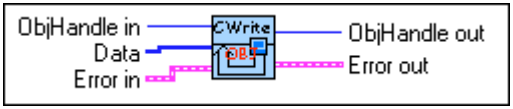
ncWrite

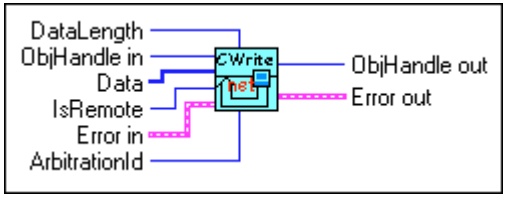
Purpose

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

ncWrite

Continued

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. In order 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 NC_ERR_OVERFLOW error (along with qualifier NC_QUAL_OVFL_WRITE), 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.

ncWrite

Continued

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 Host 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 NC_ERR_CAN_BUS_OFF 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 Value with Call), 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 attribute.

ncWrite

Continued

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.

Return Status

NC_SUCCESS	Success (no error or warning).
NC_ERR_BAD_PARAM	Invalid parameter.
NC_ERR_DRIVER	Implementation-specific error in the NI-CAN driver.
NC_ERR_OVERFLOW	Write queue overflow. This error occurs when the write queue of the object is full, and the data value you provided cannot be queued for later transmission. The error can occur only if the write queue length (NC_ATTR_WRITE_Q_LEN) is nonzero.
NC_ERR_TIMEOUT	Watchdog timeout expired for a CAN Object. This error code does not apply to ncWrite itself, but indicates an error in background communication.
NC_ERR_CAN_BUS_OFF	Error or warning indicating CAN communication errors. This error code does not apply to ncWrite itself, but indicates an error in background communication.

Examples

These examples assume the following declarations:

```
NCTYPE_STATUS            status;
NCTYPE_OBJH              objh;
NCTYPE_CAN_FRAME_TIMED   wframe;
NCTYPE_CAN_DATA_TIMED    wdata;
```

ncWrite

Continued

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);
```


Chapter 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`	You can get or set the attribute at any time.
`Config`	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. NI-CAN obtains the initial value of configuration attributes from the NI-CAN Configuration utility, and you can override them only by using `ncConfig`, not `ncSetAttribute`.

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, or PCMCIA 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 status code NC_ERR_CAN_BUS_OFF and a warning severity.

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 status code NC_ERR_CAN_BUS_OFF and an error severity.

Whether the severity of NC_ERR_CAN_BUS_OFF is a warning or error, the status qualifier is set to indicate the most recently detected communications error. This qualifier can have the value NC_QUAL_CAN_STUFF (more than five equal bits), NC_QUAL_CAN_FORM (wrong frame format), NC_QUAL_CAN_ACK (frame not acknowledged), NC_QUAL_CAN_BIT1 (transmitted one but detected zero), NC_QUAL_CAN_BIT0, or NC_QUAL_CAN_CRC (wrong CRC checksum). Refer to the CAN protocol specification for a complete description of these communication errors.

If no CAN devices are connected to the network interface port, and you attempt to transmit a frame, the NC_ERR_CAN_BUS_OFF status occurs with a warning severity. 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 NC_ERR_CAN_BUS_OFF warning 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 NC_ERR_CAN_BUS_OFF 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.

Attributes

NC_ATTR_ABS_TIME (Absolute Time)

Attribute ID	NC_ATTR_ABS_TIME
Hex Encoding	80000008
Data Type	NC_ATTR_ABS_TIME
Permissions	Set
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.

NC_ATTR_BAUD_RATE (Baud Rate)

Attribute ID	NC_ATTR_BAUD_RATE
Hex Encoding	80000007
Data Type	NCTYPE_BAUD_RATE
Permissions	Config
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
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 Host 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
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 Host 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
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
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_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_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
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_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	8000006
Data Type	NCTYPE_BOOL
Permissions	Config
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	8000009
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 NC_SUCCESS. When you read an error or warning from this attribute, NI-CAN clears the appropriate state and sets the background status back to NC_SUCCESS. 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
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
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
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 or not 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
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
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. A period of zero causes transmissions to occur in succession. 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_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
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_RX_CHANGES_ONLY (Receive Changes Only)

Attribute ID	NC_ATTR_RX_CHANGES_ONLY
Hex Encoding	80000015
Data Type	NCTYPE_BOOL
Permissions	Config
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 is always NC_SUCCESS. When you read an error or warning from this attribute, NI-CAN clears the appropriate state, and the background status is set back to NC_SUCCESS. For communication errors such as NC_ERR_CAN_BUS_OFF, 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
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).

Receive Periodically Using Remote (NC_ATTR_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).

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 NC_ERR_TIMEOUT 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 *Receive Value by Call Using Remote*, 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).

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 open 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.

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.

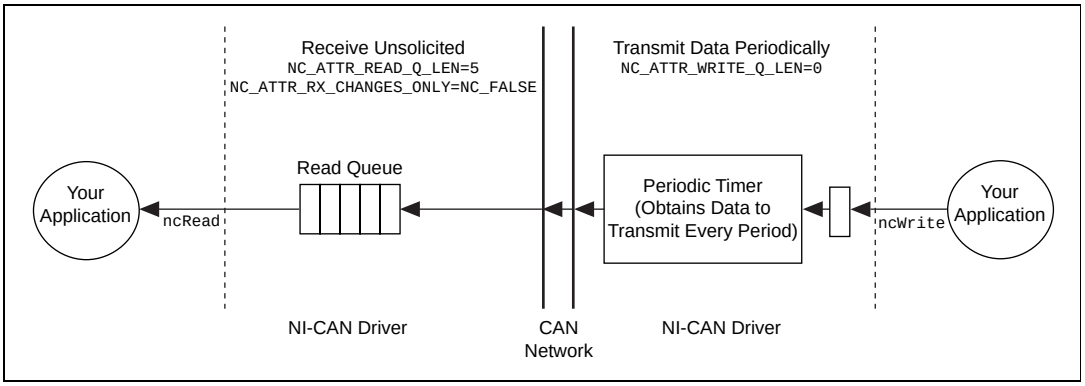


Figure 3-1. Example of Periodic Transmission

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`.

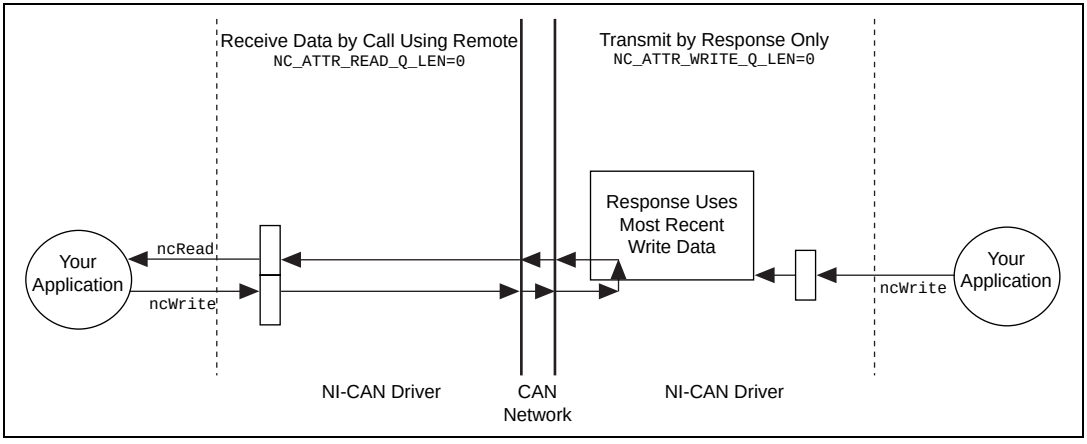


Figure 3-2. Example of Polling Remote Data Using ncWrite

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.

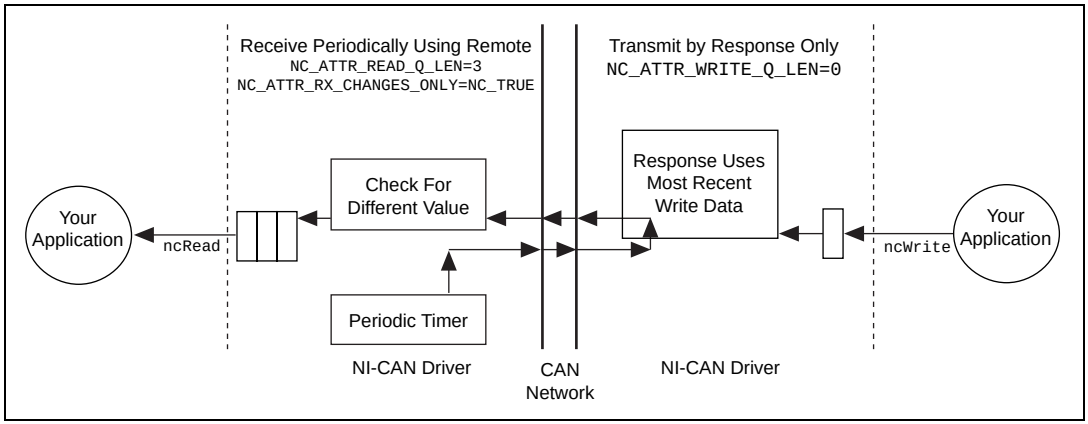


Figure 3-3. Example of Periodic Polling of Remote Data

NI-CAN Object States

Appendix
A

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	0	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.
- Use ncCreateNotification to register a callback for one or more states.

Table A-1 describes each object state.

Table A-1. NI-CAN Object States

Constant	Bitmask (Hex)	Description
NC_ST_READ_AVAIL	00000001 (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	00000002 (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 (Hex)	Description
NC_ST_STOPPED	00000004 (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_ERROR	00000010 (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	00000020 (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).

Status Codes and Qualifiers

Appendix
B

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.

NI-CAN Status Format

To provide the maximum amount of information, the status returned by NI-CAN functions is encoded as a signed 32-bit integer. The format of this integer is shown in Figure B-1.

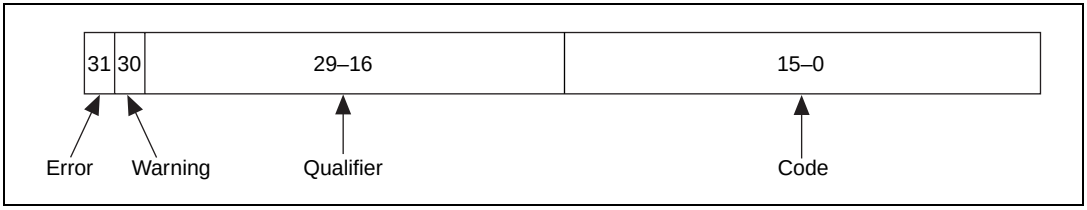


Figure B-1. Status Format

Error/Warning Indicators (Severity)

The error and warning bits ensure that all NI-CAN errors generate a negative status, and all NI-CAN warnings generate a positive status. The error bit is set when a function does not perform the expected behavior, resulting in a negative status. The warning bit is set when the function performed as expected, but a condition exists that may require your attention. If no error or warning occurs, the entire status is set to zero to indicate success. Table B-1 summarizes the behavior of NI-CAN status.

Table B-1. Determining Severity of Status

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

Code

The code bits indicate the primary status code used for warning or errors.

Qualifier

The qualifier bits hold a qualifier for the warning or error code. It is specific to individual values for the code field, and provides additional information useful for detailed debugging. For example, if the status code indicates an invalid function parameter, the qualifier holds a number which indicates the exact parameter that is invalid (one for the first parameter, two for the second, and so on). If no qualifier exists, this field has the value `NC_QUAL_NONE (0)`.

Checking Status in LabVIEW

For applications written in G (LabVIEW), status checking is basically handled automatically. For all of the 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 Reference.

In the NI-CAN implementation of Error Clusters, the `status` parameter is set to true when an error occurs, and is set to false when a warning or success occurs. The `code` parameter of the Error Cluster contains the code and qualifier fields of the NI-CAN status. If the `code` parameter of the Error Cluster is not zero, then a warning or error was detected. When the `status` parameter is true, the `source` parameter of the Error Cluster provides the name of the NI-CAN function in which the error occurred.

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.

For each NI-CAN function, you can find numeric values for the returned status code and qualifier in the online description of the function, which you can access in the Block Diagram by selecting the function and typing <Ctrl-H>.

Checking Status in C

For applications written in C or C++, you should define a function to handle NI-CAN warnings and errors. When this function detects an error, it closes all open objects, then exits the application. When this function detects a warning, it can display a warning message or simply ignore the warning. If the function has the following prototype:

```
void CheckStat(NCTYPE_STATUS stat, char *msg);
```

then your application invokes it as follows:

```
if (status != 0)
   CheckStat(status, "NI-CAN error or warning");
```

For an example implementation of the CheckStat function, refer to the C language examples in the NI-CAN examples directory.

When accessing the NI-CAN code and qualifier within your application, you should use the constants defined in nican.h. These constants have the same names as described later in this appendix. For example, to check for a timeout, you would use code such as the following:

```
if (NC_STATCODE(status) == NC_ERR_TIMEOUT)
   printf("NI-CAN timeout");
```

NI-CAN Status Codes and Qualifiers

Table B-2 summarizes each NI-CAN status code (lower 16 bits of status). After the table, a separate section for each status code lists the valid encodings for the entire status, including the associated qualifier and severity.

Table B-2. Summary of Status Codes

Code	Hex Encoding of Code (Lower 16 Bits)	Description
NC_SUCCESS	0000	Success (no warning or error)
NC_ERR_TIMEOUT	0001	Timeout Expired
NC_ERR_DRIVER	0002	Implementation-specific error in NI-CAN driver
NC_ERR_BAD_NAME	0003	Invalid or unrecognized object name
NC_ERR_BAD_PARAM	0004	Invalid function parameter
NC_ERR_BAD_VALUE	0005	Invalid attribute value
NC_ERR_ALREADY_OPEN	0006	Object already opened by another application
NC_ERR_NOT_STOPPED	0007	Attempted to set a configuration attribute while object was running
NC_ERR_OVERFLOW	0008	Queue overflow
NC_ERR_OLD_DATA	0009	Data returned from ncRead matches data returned from previous call to ncRead
NC_ERR_CAN_BUS_OFF	0101	Error or warning indicating large number of CAN communication errors

NC_SUCCESS (0000 Hex)

Success (no warning or error).

Hex Status Encoding 00000000

Qualifier	0
Severity	Success
Description	The qualifier is always zero.

NC_ERR_TIMEOUT (0001 Hex)

A timeout expired in the NI-CAN driver. The qualifier indicates the type of timeout that expired.

Hex Status Encoding 80000001

Qualifier	NC_QUAL_TIMO_FUNCTION (0)
Severity	Error
Description	The timeout of ncWaitForState or ncCreateNotification expired before any desired states occurred.
Solutions	• Increase the value of the Timeout parameter to wait longer. • If the timeout occurs while waiting for NC_ST_READ_AVAIL or NC_ST_WRITE_SUCCESS, verify your CAN cable connections, and ensure that remote devices are operating properly. • If you wait only for a background error or warning, the timeout is often the expected behavior, and you can ignore it.

Hex Status Encoding 80010001

Qualifier	NC_QUAL_TIMO_WATCHDOG (1)
Severity	Error
Description	The watchdog timeout for a CAN Object expired, indicating that data was not received at the rate expected. This error occurs in the background and is returned by ncRead and ncWrite.
Solutions	• Verify your CAN cable connections, and ensure that remote devices are operating properly. • If the remote device takes longer than expected to transmit data, you can increase the period specified in the NC_ATTR_BKD_PERIOD attribute.

NC_ERR_DRIVER (0002 Hex)

An implementation-specific error has occurred in the NI-CAN driver, such as the inability to allocate needed memory. This error should never occur under normal circumstances.

Hex Status Encoding 8xxx0002, 9xxx0002, Axxx0002, and Bxxx0002

Qualifier	Varies
Severity	Error
Description	The qualifier (bits 16-29) holds a value that is specific to the NI-CAN driver implementation.
Solutions	Write down the status value, and contact National Instruments for technical support.

NC_ERR_BAD_NAME (0003 Hex)

The `ObjName` parameter of `ncOpenObject` or `ncConfig` contains an invalid or unrecognized name.

Hex Status Encoding 80000003

Qualifier	`0`
Severity	Error
Description	There is a basic syntax error such as an invalid character or a single colon instead of a double colon.
Solutions	• Verify that the object name does not contain invalid characters, and that you use the syntax specified in `ncOpenObject`. • If you are opening a user-defined alias, use the NI-CAN Configuration utility to verify that the alias is defined in the list of CAN Objects.

Hex Status Encoding 80010003

Qualifier	`1`
Severity	Error
Description	The CAN Network Interface Object name is invalid or unknown.
Solutions	Use the NI-CAN Configuration utility to verify that the CAN Network Interface Object is assigned a physical CAN port. The NI-CAN Diagnostic utility also provides a list of valid CAN Network Interface Object names.

Hex Status Encoding 80020003

Qualifier	1
Severity	Error
Description	The CAN Object name is invalid or unknown.
Solutions	Verify that you use the syntax specified in the *CAN Object* section of Chapter 3, *NI-CAN Objects*.

NC_ERR_BAD_PARAM (0004 Hex)

A function parameter is invalid.

Hex Status Encoding 800x0004

Qualifier	Varies
Severity	Error
Description	The qualifier holds the position of the invalid parameter in the C function prototype. For example, if the `DataSize` parameter of `ncRead` is invalid, the qualifier is two (status 80020004).
Solutions	Check the qualifier, then read the function description in Chapter 2, *NI-CAN Functions*, to verify that you provide a valid value for the specified parameter.

NC_ERR_BAD_VALUE (0005 Hex)

The attribute value for the specified attribute ID is invalid. For example, if you call `ncSetAttribute` with the `AttrId NC_ATTR_BAUD_RATE`, and `AttrPtr` points to an invalid baud rate such as 20005, `NC_ERR_BAD_VALUE` is returned.

Hex Status Encoding 80000005

Qualifier	`0` (for `ncSetAttribute`)
Severity	Error
Description	For `ncSetAttribute`, the qualifier is always zero.
Solutions	Check the description of the attribute in Chapter 3, *NI-CAN Objects*, and verify that the value you pass is valid.

Hex Status Encoding 8xxx0005

Qualifier	Varies (for `ncAction` and `ncConfig`)
Severity	Error
Description	For `ncAction` and `ncConfig`, this error indicates that although each configuration attribute holds a valid value, the combination of values is invalid. For example, if a CAN Object is configured as Transmit Value Periodically, the period attribute must be nonzero. For this error, the qualifier holds the low order bits of the `AttrId` of one of the invalid attributes.
Solutions	Using the attribute ID provided in the qualifier, check the description of the attribute in Chapter 3, *NI-CAN Objects*, and verify that the value you set works with the other attribute values.

NC_ERR_ALREADY_OPEN (0006 Hex)

The object has already been opened by another application. If one application opens an object, no other application can open or configure that object until the object is closed.

Hex Status Encoding 80000006

Qualifier	`0`
Severity	Error
Description	The qualifier is always zero.
Solutions	• If you have two or more applications that open the same object, run only one application at a time. • If two or more applications need to share an object, you can alternate access by closing the object in one application, then opening the object in another. • Before exiting your application, verify that you call `ncCloseObject` for every object opened. For LabVIEW, you should implement a control on your front panel to stop the program and close all objects. You should not use the LabVIEW **Stop** button to stop execution, because doing so often prevents proper use of `ncCloseObject`

NC_ERR_NOT_STOPPED (0007 Hex)

You attempted to set a configuration attribute for an object while the object was running. You can change attributes with `Config` permissions only when the object is stopped (not communicating).

Hex Status Encoding 80000007

Qualifier	`0`
Severity	Error
Description	The qualifier is always zero.
Solutions	• Configure the object prior to opening it, either within the NI-CAN Configuration utility, or by using `ncConfig`. • Use `ncAction` to stop and start communication as needed so that you can update configuration attributes.

NC_ERR_OVERFLOW (0008 Hex)

There is a queue overflow.

Hex Status Encoding 80000008

Qualifier	NC_QUAL_OVFL_WRITE
Severity	Error
Description	There is a write queue overflow. This error occurs when you call ncWrite for a full write queue. It occurs only when the length of the write queue is greater than zero.
Solutions	• Increase the length of the write queue using the NC_ATTR_WRITE_Q_LEN attribute. • Prior to calling ncWrite, check NC_ATTR_WRITE_PENDING to verify that it is less than the write queue length. • If you merely want the most recent data to be transmitted, such as for periodic transmission, set NC_ATTR_WRITE_Q_LEN to zero. • Wait for the NC_ST_WRITE_SUCCESS state before calling ncWrite to queue more data.

Hex Status Encoding 80010008

Qualifier	NC_QUAL_OVFL_READ
Severity	Error
Description	There is a read queue overflow. This error occurs when new data is received from the network for a full read queue, and NI-CAN discards it. The error occurs only when the length of the read queue is greater than zero. This error occurs in the background, and is returned by ncRead and ncWrite.
Solutions	• Increase the length of the read queue using the NC_ATTR_READ_Q_LEN attribute. • Call ncRead more often in your application. One way to do this is to create a notification for NC_ST_READ_AVAIL using ncCreateNotification, so that you can read data as soon as it becomes available. • If you merely want the most recent data from ncRead, set NC_ATTR_READ_Q_LEN to zero. • Check NC_ATTR_READ_PENDING for a given threshold prior to calling ncRead.

Hex Status Encoding 80020008

Qualifier	NC_QUAL_OVFL_CHIP
Severity	Error
Description	There is an overflow in the CAN communications controller chip. This error occurs in the background and is returned by ncRead and ncWrite.
Solutions	Disable timestamping by setting the NC_ATTR_TIMESTAMPING attribute to NC_FALSE.

NC_ERR_OLD_DATA (0009 Hex)

The data returned from ncRead matches the data returned from the previous call to ncRead. Because the old data is returned successfully, this status code has a warning severity, not error.

Hex Status Encoding 40000009

Qualifier	0
Severity	Warning
Description	The qualifier is always zero.
Solutions	• If you merely want to read the most recent data, ignore this warning. • Wait for the NC_ST_READ_AVAIL state before calling ncRead.

NC_ERR_CAN_BUS_OFF (0101 Hex)

This is an error or warning that can indicate many different CAN communication errors. When the transmit or receive error counter of the CAN communications controller chip increments above 96, a warning occurs. When the transmit error counter increments above 255 (bus off), an error occurs and the network interface is stopped. In both cases the qualifier is set to the most recent detected communication error. This warning/error occurs in the background, and is returned by ncRead and ncWrite. For more information, refer to the *CAN Network Interface Object* section of Chapter 3, *NI-CAN Objects*.

The solutions for all of the qualifiers of the NC_ERR_CAN_BUS_OFF error follow the descriptions.

Hex Status Encoding 40010101 and 80010101

Qualifier	NC_QUAL_CAN_STUFF
Severity	Varies
Description	A stuff error has occurred (more than five equal bits in the frame).

Hex Status Encoding 40020101 and 80020101

Qualifier	NC_QUAL_CAN_FORM
Severity	Varies
Description	The frame format is wrong.

Hex Status Encoding 40030101 and 80030101

Qualifier	NC_QUAL_CAN_ACK
Severity	Varies
Description	The frame has not been acknowledged.

Hex Status Encoding 40040101 and 80040101

Qualifier	NC_QUAL_CAN_BIT1
Severity	Varies
Description	One was transmitted but zero was detected.

Hex Status Encoding 40050101 and 80050101

Qualifier	NC_QUAL_CAN_BIT0
Severity	Varies
Description	Zero was transmitted but one was detected.

Hex Status Encoding 40060101 and 80060101

Qualifier	NC_QUAL_CAN_CRC
Severity	Varies
Description	The CRC checksum is invalid.

Solutions

The following solutions apply to all of the qualifiers for the NC_ERR_CAN_BUS_OFF error:

- CAN communication errors are often caused by defective cabling. Verify that your connector, cables, and devices are functioning properly.
- If you attempt to transmit a CAN frame without another CAN device connected, or with the bus powered off, the NC_ERR_CAN_BUS_OFF warning occurs. Connect your other CAN devices prior to attempting communication.

Customer Communication

Appendix
C

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

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

Electronic Services

Bulletin Board Support

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

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

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

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

FTP Support

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

Fax-on-Demand Support

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

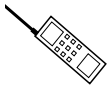
E-Mail Support (currently U.S. only)

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

`support@natinst.com`

Telephone and Fax Support

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



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

Technical Support Form

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

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

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___) _____

Computer brand _____ Model _____ Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem:_____

_____

_____

_____

_____

_____

Documentation Comment Form

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

Title: *NI-CAN™ Programmer Reference Manual for Win32*

Edition Date: November 1997

Part Number: 321369B-01

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

_____

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (___)_____ Fax (___) _____

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

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

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 Sex 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 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, Mary 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.

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

L

M

N

Q

R

S

T