CAN

NI-CAN™ User Manual

August 1999 Edition
Part Number 321370C-01

Worldwide Technical Support and Product Information

www.natinst.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00, Singapore 2265886,
Spain (Barcelona) 93 582 0251, Spain (Madrid) 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

BridgeVIEW™, CVI™, LabVIEW™, `natinst.com`™, National Instruments™, and NI-CAN™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein 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 for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure, or by errors on the part of the user or application designer. Because each end-user system is customized and differs from National Instruments testing platforms and because a user or application designer may use National Instruments products in combination with other products in a manner not evaluated or contemplated by National Instruments, the user or application designer is ultimately responsible for verifying and validating the suitability of National Instruments products whenever National Instruments products are incorporated in a system or application, including, without limitation, the appropriate design, process and safety level of such system or application.

Contents

About This Manual

Chapter 1
Introduction

Chapter 2
Developing Your Application

Chapter 3
NI-CAN Programming Techniques

Chapter 4
Application Examples

Chapter 5
NI-CAN Configuration Utility

Appendix A
Uninstalling the Hardware and Software

Appendix B
Windows 98/95: Troubleshooting and Common Questions

Appendix C
Windows NT: Troubleshooting and Common Questions

Appendix D
Technical Support Resources

Glossary

Index

Figures

Table

About This Manual

This manual describes the features of the NI-CAN software. 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 the NI-CAN software.

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

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

Conventions

The following conventions appear 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** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

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

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

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

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories,

programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

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*
- ANSI/ISO Standard 11519-1, 2 *Road Vehicles—Low Speed Serial Data Communications*, Part 1 and 2
- *CAN Specification Version 2.0*, 1991, Robert Bosch Gmbh., Postfach 500, D-7000 Stuttgart 1
- LabVIEW Online Reference
- Win32 Software Development Kit (SDK) online help
- *Microsoft User's Guide*

1

Introduction

This chapter gives an overview of CAN and the NI-CAN software.

CAN Overview

History and Usage of CAN

In the past few decades, the need for improvements in automotive technology has led to increased usage of electronic control systems for functions such as engine timing, anti-lock brake systems, and distributorless ignition. With conventional wiring, data is exchanged in these systems using dedicated signal lines. As the complexity and number of devices has increased, usage of dedicated signal lines has become increasingly difficult and expensive.

To overcome the limitations of conventional automotive wiring, Bosch developed the Controller Area Network (CAN) in the mid-1980s. Using CAN, devices (controllers, sensors, and actuators) are connected on a common serial bus. This network of devices can be thought of as a scaled down, real-time, low cost version of networks used to connect personal computers. Any device on a CAN network can communicate with any other device using a common pair of wires.

As CAN implementations increased in the automotive industry, CAN was standardized internationally as ISO 11898, and CAN chips were created by major semiconductor manufacturers such as Intel, Motorola, and Phillips. With these developments, many manufacturers of industrial automation equipment began to consider CAN for usage in industrial applications. Comparison of the requirements for automotive and industrial device networks showed many similarities, including the transition away from dedicated signal lines, low cost, resistance to harsh environments, and high real-time capabilities.

Because of these similarities, CAN became widely used in industrial applications such as textile machinery, packaging machines, and production line equipment such as photoelectric sensors and motion

controllers. By the mid-1990s, CAN was specified as the basis of many industrial device networking protocols, including DeviceNet, CANopen, and Smart Distributed System (SDS).

With its growing popularity in automotive and industrial applications, CAN has been increasingly used in a wide variety of diverse applications. Usage in systems such as agricultural equipment, nautical machinery, medical apparatus, semiconductor manufacturing equipment, and machine tools testify to the incredible versatility of CAN.

CAN Identifiers and Message Priority

When a CAN device transmits data onto the network, an identifier that is unique throughout the network precedes the data. The identifier defines not only the content of the data, but also the priority. A CAN identifier, along with its associated data, is often referred to as a *CAN Object*.

When a device transmits a message onto the CAN network, all other devices on the network receive that message. Each receiving device performs an acceptance test on the identifier to determine if the message is relevant to it. If the received identifier is not relevant to the device (such as RPM received by an air conditioning controller), the device ignores the message.

When more than one CAN device transmits a message simultaneously, the identifier is used as a priority to determine which device gains access to the network. The lower the numerical value of the identifier, the higher its priority.

Figure 1-1 shows two CAN devices attempting to transmit messages, one using identifier 647 hex, and the other using identifier 6FF hex. As each device transmits the 11 bits of its identifier, it examines the network to determine if a higher-priority identifier is being transmitted simultaneously. If an identifier collision is detected, the losing device(s) immediately cease transmission, and wait for the higher-priority message to complete before automatically retrying. Because the highest priority identifier continues its transmission without interruption, this scheme is referred to as *non-destructive bitwise arbitration*, and CAN's identifier is often referred to as an *arbitration ID*. This ability to resolve collisions and continue with high-priority transmissions is one feature that makes CAN ideal for real-time applications.

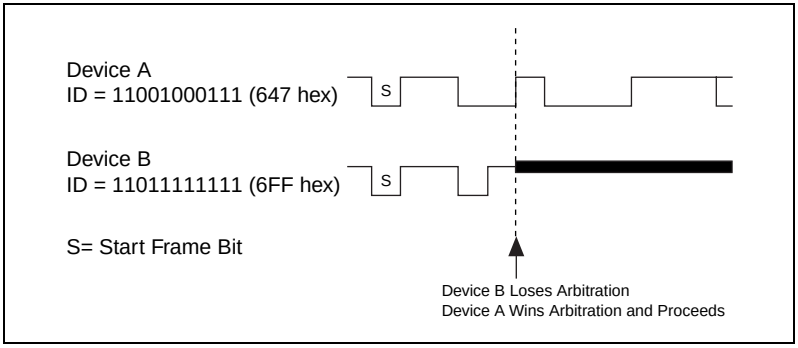

Figure 1-1. Example of CAN Arbitration

CAN Frames

In a CAN network, the messages transferred across the network are called frames. The CAN protocol supports two frame formats as defined in the Bosch version 2.0 specifications, the essential difference being in the length of the arbitration ID. In the standard frame format (also known as 2.0A), the length of the ID is 11 bits. In the extended frame format (also known as 2.0B), the length of the ID is 29 bits. The ISO 11898 specification supports only the standard frame format. Figure 1-2 shows the essential fields of the standard and extended frame formats, and the following sections describe each field.

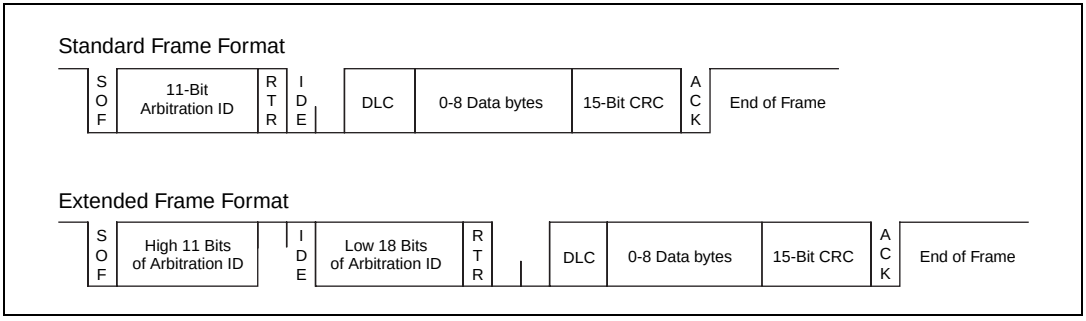

Figure 1-2. Standard and Extended Frame Formats

Start of Frame (SOF)

Start of Frame is a single bit (0) that marks the beginning of a CAN frame.

Arbitration ID

The arbitration ID fields contain the identifier for a CAN frame. The standard format has one 11-bit field, and the extended format has two fields, which are 11 and 18 bits in length. In both formats, bits of the arbitration ID are transmitted from high to low order.

Remote Transmit Request (RTR)

The Remote Transmit Request bit is dominant (0) for data frames, and recessive (1) for remote frames. Data frames are the fundamental means of data transfer on a CAN network, and are used to transmit data from one device to one or more receivers. A device transmits a remote frame to request transmission of a data frame for the given arbitration ID. The remote frame is used to request data from its source device, rather than waiting for the data source to transmit the data on its own.

Identifier Extension (IDE)

The Identifier Extension bit differentiates standard frames from extended frames. Because the IDE bit is dominant (0) for standard frames and recessive (1) for extended frames, standard frames are always higher priority than extended frames.

Data Length Code (DLC)

The Data Length Code is a 4-bit field that indicates the number of data bytes in a data frame. In a remote frame, the Data Length Code indicates the number of data bytes in the requested data frame. Valid Data Length Codes range from zero to eight.

Data Bytes

For data frames, this field contains from 0 to 8 data bytes. Remote CAN frames always contain zero data bytes.

Cyclic Redundancy Check (CRC)

The 15-bit Cyclic Redundancy Check detects bit errors in frames. The transmitter calculates the CRC based on the preceding bits of the frame, and all receivers recalculate it for comparison. If the CRC calculated by a receiver differs from the CRC in the frame, the receiver detects an error.

Acknowledgment Bit (ACK)

All receivers use the Acknowledgment Bit to acknowledge successful reception of the frame. The ACK bit is transmitted recessive (1), and is overwritten as dominant (0) by all devices that receive the frame successfully. The receivers acknowledge correct frames regardless of the acceptance test performed on the arbitration ID. If the transmitter of the frame detects no acknowledgment, it could mean that the receivers detected an error (such as a CRC error), the ACK bit was corrupted, or there are no receivers (for example, only one device on the network). In such cases, the transmitter automatically retransmits the frame.

End of Frame

Each frame ends with a sequence of recessive bits. After the required number of recessive bits, the CAN bus is idle, and the next frame transmission can begin.

CAN Error Detection and Confinement

One of the most important and useful features of CAN is its high reliability, even in extremely noisy environments. CAN provides a variety of mechanisms to detect errors in frames. This error detection is used to retransmit the frame until it is received successfully. CAN also provides an error confinement mechanism used to remove a malfunctioning device from the CAN network when a high percentage of its frames result in errors. This error confinement prevents malfunctioning devices from disturbing the overall network traffic.

Error Detection

Whenever any CAN device detects an error in a frame, that device transmits a special sequence of bits called an error flag. This error flag is normally detected by the device transmitting the invalid frame, which then retransmits to correct the error. The retransmission starts over from the start of frame, and thus arbitration with other devices is again possible.

CAN devices detect the following errors, which are described in the following sections:

- Bit error
- Stuff error
- CRC error
- Form error
- Acknowledgment error

Bit Error

During frame transmissions, a CAN device monitors the bus on a bit-by-bit basis. If the bit level monitored is different from the transmitted bit, a bit error is detected. This bit error check applies only to the Data Length Code, Data Bytes, and Cyclic Redundancy Check fields of the transmitted frame.

Stuff Error

Whenever a transmitting device detects five consecutive bits of equal value, it automatically inserts a complemented bit into the transmitted bit stream. This stuff bit is automatically removed by all receiving devices. The bit stuffing scheme is used to guarantee enough edges in the bit stream to maintain synchronization within a frame.

A stuff error occurs whenever six consecutive bits of equal value are detected on the bus.

CRC Error

A CRC error is detected by a receiving device whenever the calculated CRC differs from the actual CRC in the frame.

Form Error

A form error occurs when a violation of the fundamental CAN frame encoding is detected. For example, if a CAN device begins transmitting the Start Of Frame bit for a new frame before the End Of Frame sequence completes for a previous frame (does not wait for bus idle), a form error is detected.

Acknowledgment Error

An acknowledgment error is detected by a transmitting device whenever it does not detect a dominant Acknowledgment Bit (ACK).

Error Confinement

To provide for error confinement, each CAN device must implement a transmit error counter and a receive error counter. The transmit error counter is incremented when errors are detected for transmitted frames, and decremented when a frame is transmitted successfully. The receive error counter is used for received frames in much the same way. The error counters are increased more for errors than they are decreased for successful reception/transmission. This ensures that the error counters will generally increase when a certain ratio of frames (roughly 1/8) encounter

errors. By maintaining the error counters in this manner, the CAN protocol can generally distinguish temporary errors (such as those caused by external noise) from permanent failures (such as a broken cable). For complete information on the rules used to increment/decrement the error counters, refer to the CAN specification (ISO 11898).

With regard to error confinement, each CAN device may be in one of three states: error active, error passive, and bus off.

Error Active State

When a CAN device is powered on, it begins in the error active state. A device in error active state can normally take part in communication, and transmits an active error flag when an error is detected. This active error flag (sequence of dominant 0 bits) causes the current frame transmission to abort, resulting in a subsequent retransmission. A CAN device remains in the error active state as long as the transmit and receive error counters are both below 128. In a normally functioning network of CAN devices, all devices are in the error active state.

Error Passive State

If either the transmit error counter or the receive error counter increments above 127, the CAN device transitions into the error passive state. A device in error passive state can still take part in communication, but transmits a passive error flag when an error is detected. This passive error flag (sequence of recessive 1 bits) generally does not abort frames transmitted by other devices. Since passive error flags are not able to prevail over any activity on the bus line, they are noticed only when the error passive device is transmitting a frame. Thus, if an error passive device detects a receive error on a frame which is received successfully by other devices, the frame is not retransmitted.

One special rule to keep in mind is that when an error passive device detects an acknowledgment error, it does not increment its transmit error counter. Thus, if a CAN network consists of only one device (for instance, if you do not connect a cable to your National Instruments CAN interface), and that device attempts to transmit a frame, it retransmits continuously but never goes into bus off state (although it eventually reaches error passive state).

Bus Off State

If the transmit error counter increments above 255, the CAN device transitions into the bus off state. A device in the bus off state does not transmit or receive any frames, and thus cannot have any influence on the

bus. The bus off state is used to disable a malfunctioning CAN device which frequently transmits invalid frames, so that the device does not adversely impact other devices on the network. When a CAN device has transitioned to bus off, it can only be placed back into error active state (with both counters reset to zero) by manual intervention. For sensor/actuator types of devices, this often involves powering the device off then on. For NI-CAN network interfaces, communication can be started again using a function such as `ncAction`.

Low-Speed CAN

Low-speed CAN is commonly used to control "comfort" devices in an automobile, such as seat adjustment, mirror adjustment, and door locking. It differs from "high-speed" CAN in that the maximum baud rate is 125K and it utilizes CAN transceivers that offer fault-tolerant capability. This enables the CAN bus to keep operating even if one of the wires is cut or short-circuited because it operates on relative changes in voltage, and thus provides a much higher level of safety. The fault tolerance feature means that communications capability is maintained even if any of the ISO11519 wiring failures occur. The transceiver solves many common and frequent wiring problems such as poor connectors, and also overcomes short circuits of either transmission wire to ground or battery voltage, or the other transmission wire. The transceiver resolves the fault situation without involvement of external hardware or software. On the detection of a fault, the transceiver switches to a one wire transmission mode and automatically switches back to differential mode if the fault is removed.

Special resistors are added to the circuitry for the proper operation of the fault-tolerant transceiver. The values of the resistors depend on the number of nodes and the resistance values per node. For guidelines on selecting the resistor, refer to the cabling appendix of your NI-CAN getting started manual.

Because the low-speed transceiver switches to a fault tolerant mode on fault detection and continues to maintain communications, NI-CAN provides a special attribute, `NC_ATTR_LOG_COMM_ERRS`, which when set to `NC_TRUE` enables the reporting of such errors in the Read queue of the Network interface rather than in the Status returned from a function call. The default value of this attribute is `NC_FALSE` (for high-speed interface). Refer to the CAN network interface object attributes section in the *NI-CAN Programmer Reference Manual* for details on how to use this attribute.

NI-CAN Software Overview

Independent Design

The NI-CAN Application Programming Interface (API), like most National Instruments APIs, is largely independent of operating system and programming language. You can use NI-CAN in a wide variety of programming environments, including LabVIEW and C programming environments such as LabWindows/CVI. Applications written for NI-CAN are also portable across different operating systems, such as Windows 98/95 and Windows NT.

In addition to being independent of operating system and programming language, NI-CAN is designed to be largely independent of a specific device network protocol. Device network independence means that where possible, terminology specific to CAN alone is avoided so that the API can be expanded later to support higher level protocols based on CAN. Examples of such protocols include DeviceNet, Smart Distributed System (SDS), and CANopen. Device network independence largely applies to terminology such as function names, and in no way limits access to the CAN network. For example, the function provided to read data from a CAN frame is called `ncRead`, as opposed to a name specific to CAN, such as `ncReadCanFrame`.

Object-Oriented Design

NI-CAN often uses object-oriented terminology and concepts. Object-oriented terminology provides an excellent model for describing device networks in terms that are easy to understand.

In object-oriented terminology, the term *class* describes a classification of an object, and the term *instance* refers to a specific instance of that object. The term *object* is generally used as a synonym for instance. For example, NI-CAN defines a class called the CAN Network Interface Object, which encapsulates any network interface port on a National Instruments CAN hardware product. Specific instances of the CAN Network Interface Object are referenced with names like `CAN0` and `CAN1`. Each instance of a particular class has *attributes* that define its externally visible qualities, as well as *methods* that are used to perform actions. For example, each instance of the CAN Network Interface Object has an attribute for the baud rate (bits per second) used for communication, as well as a method used to transmit CAN frames onto the network.

For more information on object-oriented and CAN terminology, refer to the *Glossary*.

NI-CAN Object Hierarchy

The basic model of the NI-CAN software architecture is a hierarchical collection of objects (instances), each of which has attributes and methods. The hierarchy shows relationships between various objects. In general, a given object in the hierarchy has an "is used to access" relationship with all objects above it in the hierarchy.

As an example, consider a CAN device network in which the network interface of a host computer is physically connected to two devices, a pushbutton and an LED, as shown in Figure 1-3.

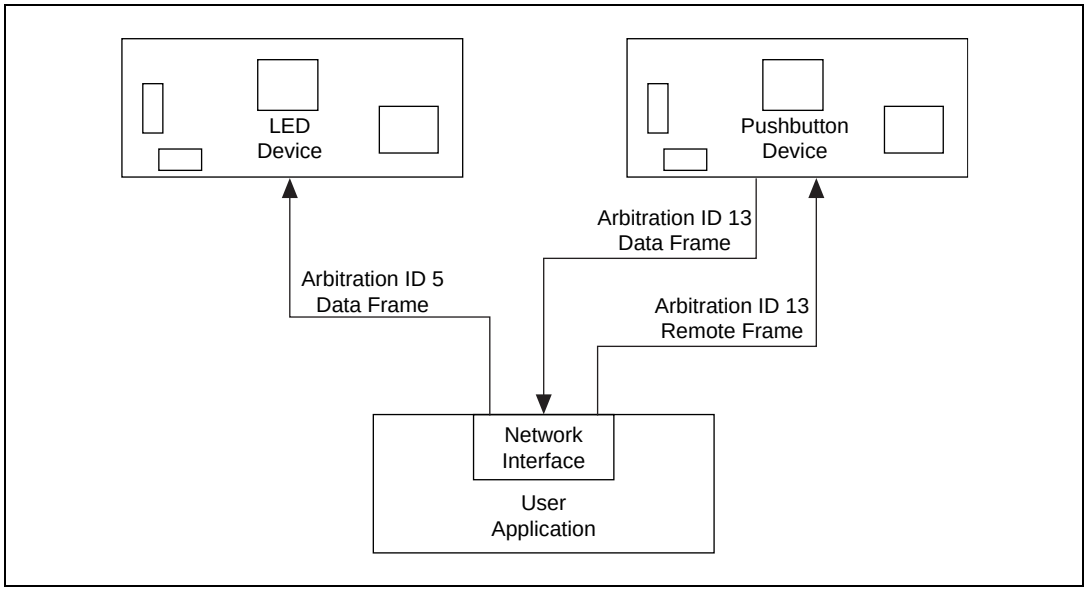

Figure 1-3. Simple CAN Device Network Application

The pushbutton device transmits the state of the button in a CAN data frame with standard arbitration ID 13. The frame data consists of a single byte—zero if the button is off, one if the button is on. For an NI-CAN application to obtain the current state of the pushbutton, it transmits a CAN remote frame with standard arbitration ID 13. The pushbutton device responds to this remote transmission request by transmitting the button state in its CAN data frame.

The LED device expects to obtain the state of the LED from a CAN data frame with standard arbitration ID 5. It expects the frame data to consist of a single byte—zero to turn the light off, one to turn the light on.

Figure 1-4 shows how NI-CAN objects encapsulate access to this CAN device network. The ovals in Figure 1-4 indicate NI-CAN objects, and the dotted lines indicate what each object encapsulates.

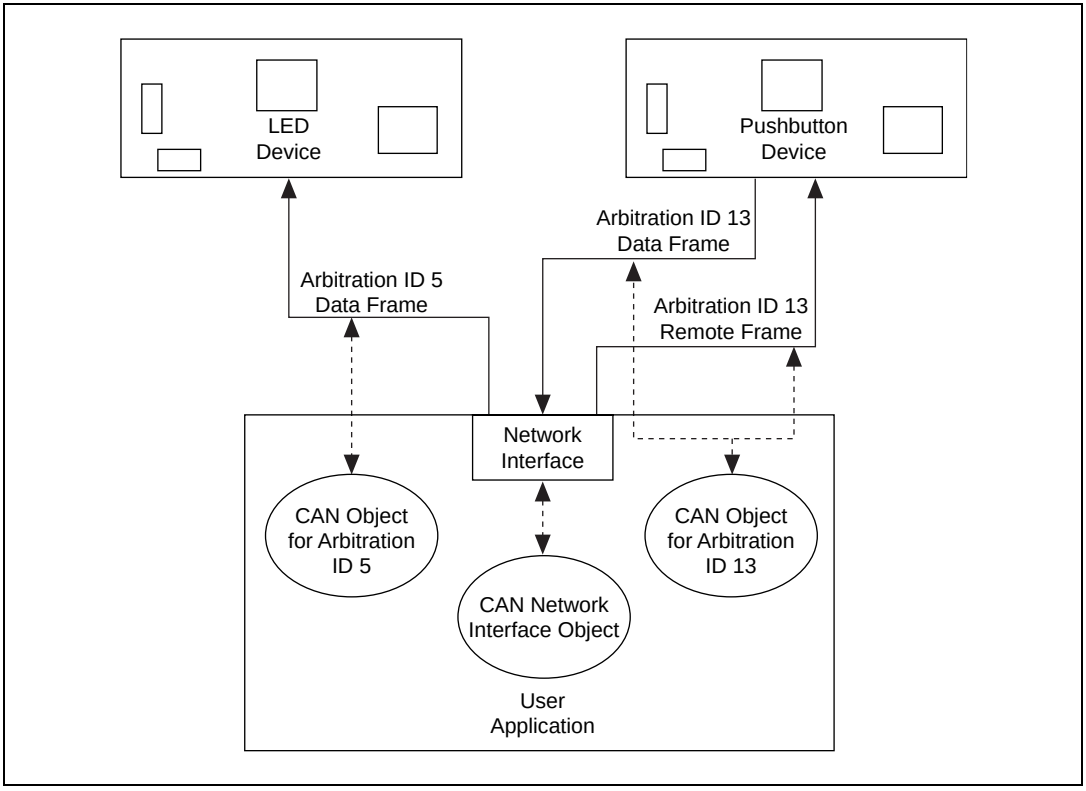

Figure 1-4. Applying NI-CAN Objects to the Example in Figure 1-3

The CAN Network Interface Object encapsulates the entire CAN network interface. Its attributes are used to configure settings that apply to the network interface as a whole. For example, the CAN Network Interface Object contains an attribute you can use to set the baud rate that the network interface hardware uses for communication. You can also use the CAN Network Interface Object to communicate on the CAN device network. For example, you can use the write function to transmit a CAN remote frame to the pushbutton device, then use the read function to retrieve the resulting CAN data frame. Because the CAN Network Interface Object provides

direct access to the network interface, the write and read functions require all information about the CAN frame to be specified, including arbitration ID, whether the frame is a CAN data frame or a CAN remote frame, the number of data bytes, and the frame data (assuming a CAN data frame).

The CAN Object encapsulates a specific arbitration ID, along with its data. In addition to providing the ability to transmit and receive frames for a specific arbitration ID, CAN Objects also provide various forms of background access. For example, you can configure a CAN Object for arbitration ID 13 (the pushbutton) to automatically transmit a CAN remote frame every 500 ms, and to store the data of the resulting CAN data frame for later retrieval. After the CAN Object is configured in this manner, you can use the read function to obtain a single data byte that holds the most recent state of the pushbutton.

NI-CAN Software Components

The following section highlights important components of the NI-CAN software, and describes the function of each component.

NI-CAN Driver and Utilities

- A documentation file, `readme.txt`, contains important information about the NI-CAN software and a description of any new features. Before you use the software, read this file for the most recent information.
- A 32-bit, multitasking aware device driver is used to interface with National Instruments CAN hardware. Under Windows 98/95, this is a dynamically loadable, Plug and Play aware virtual device driver (VxD). Under Windows NT, this is a native Windows NT kernel driver.
- A Win32 dynamic link library, `nican.dll`, acts as the interface between all CAN applications and the NI-CAN device driver.
- The NI-CAN Configuration utility is used to modify the configuration of the NI-CAN software. Under Windows 98/95, this utility is integrated into the Windows Device Manager. Under Windows NT, this utility is a control panel application.
- The NI-CAN Diagnostic utility is used to verify that the CAN hardware and software have been installed properly.

Firmware Image Files

All National Instruments CAN hardware products contain an on-board microprocessor. This microprocessor is used so that all time-critical aspects of the NI-CAN software can be executed separately from your Windows application. The firmware image which runs on the on-board microprocessor, `nican.nfw`, is loaded and executed automatically when your NI-CAN application starts up.

Language Interface Files

- A documentation file, `readme.txt`, contains information about the NI-CAN language interface files.
- A 32-bit C language include file, `nican.h`, contains NI-CAN function prototypes, host data types, and various predefined constants.
- A 32-bit C language interface file, `nicanmsc.lib`, is used by Microsoft C/C++ applications to access the NI-CAN DLL.
- A 32-bit C language interface file, `nicanbor.lib`, is used by Borland C/C++ (5.0 or greater) applications to access the NI-CAN DLL.
- A 32-bit C language interface file, `nican.lib`, is used by LabWindows/CVI applications to access the NI-CAN DLL.
- NI-CAN function panels for LabWindows/CVI (`nican.fp`) enable you to develop a CAN application with LabWindows/CVI.
- A 32-bit G function library, `nican.llb`, is used by LabVIEW applications to access the NI-CAN DLL.

Application Examples

The NI-CAN software includes three sample applications. For a detailed description of the sample application files, refer to Chapter 4, *Application Examples*.

Interaction of Software Components with Your Application

Figure 1-5 shows the interaction between your application and the NI-CAN software components.

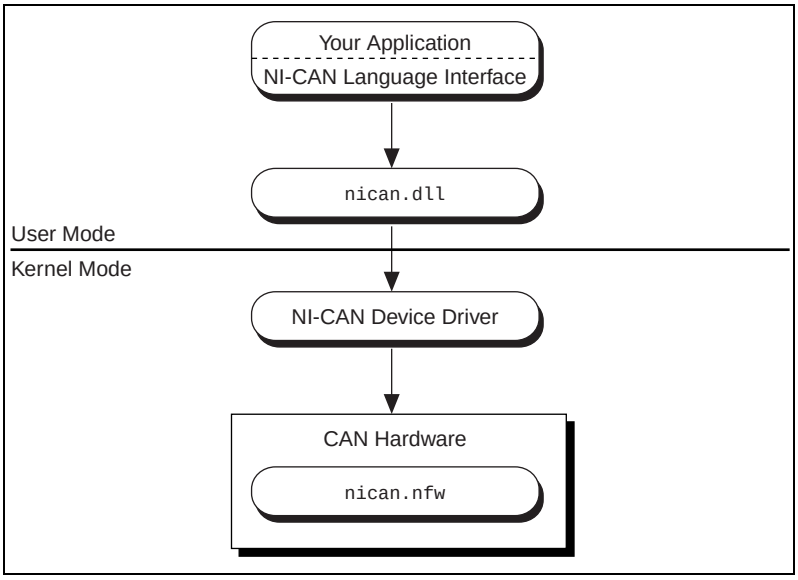

Figure 1-5. Interaction of NI-CAN Software Components

2

Developing Your Application

This chapter explains how to develop an application program using the NI-CAN functions.

Choosing Your Programming Method

Choosing a Method to Access the NI-CAN Software

Applications can access the NI-CAN dynamic link library (`nican.dll`) either by using an NI-CAN language interface or by direct entry access.

G Language (LabVIEW) Function Library

For applications written in LabVIEW (4.0 or later) or BridgeVIEW, the NI-CAN function library for G (`nican.llb`) provides a LabVIEW function to access each NI-CAN function easily.

You can add NI-CAN functions and controls to your LabVIEW palettes by selecting Select Palette Set from the LabVIEW **Edit** menu, then selecting `nican_view`. The NI-CAN functions can then be found in your LabVIEW **Functions** palette for placement into your diagram. The NI-CAN controls can be found in your LabVIEW **Controls** palette for placement into your front panel.

For a description of how each NI-CAN function in LabVIEW maps to the corresponding C language NI-CAN function, refer to the *NI-CAN Programmer Reference Manual*.

C/C++ Language Interfaces

You can use an NI-CAN C language interface if your application is written in Microsoft Visual C/C++ (2.0 or later), Borland C/C++ (5.0 or later), or LabWindows/CVI (4.0 or later) with Microsoft C. For other C/C++ compilers, you must access `nican.dll` directly.

To use a C/C++ language interface, include the `nican.h` header file in your code, then link the appropriate NI-CAN language interface file with your application. You can then call the NI-CAN functions without any extra effort.

For C applications (files with `.c` extension), include `nican.h` by adding the following line to the beginning of your code:

```
#include "nican.h"
```

For C++ applications (files with `.cpp` extension), include `nican.h` by adding the following lines to the beginning of your code:

```
#define _cplusplus
#include "nican.h"
```

The `_cplusplus` define allows `nican.h` to properly handle the transition from C++ to the C language NI-CAN functions.

For Microsoft Visual C++, link your application with the NI-CAN language interface for Microsoft C/C++, `nicanmsc.lib`.

For Borland C/C++ (5.0 or later), link your application with the NI-CAN language interface for Borland C/C++, `nicanbor.lib`. For Borland C/C++ 4.5, you must use direct entry access for NI-CAN.

For LabWindows/CVI, your application is linked with the NI-CAN language interface for LabWindows/CVI, `nican.lib`. This library is installed automatically based on the installed compatible compiler.

For detailed information on how to compile and link your NI-CAN application, refer to the `readme.txt` file in the NI-CAN `examples` directory.

Direct Entry Access

You can directly access `nican.dll` from any programming environment that allows you to request addresses of functions that a DLL exports.

To use direct entry access, you must first load `nican.dll`. The following C language code fragment illustrates how to call the Win32 `LoadLibrary` function and check for an error:

```
#include <windows.h>
#include "nican.h"

HINSTANCE NicanLib = NULL;
```

```
NicanLib=LoadLibrary("nican.dll");
if (NicanLib == NULL) {
   return FALSE;
}
```

Next, your application must use the Win32 `GetProcAddress` function to get the addresses of the NI-CAN functions your application needs to use. For each NI-CAN function used by your application, you must define a direct entry prototype. For the prototypes for each function exported by `nican.dll`, refer to the *NI-CAN Programmer Reference Manual*. The following code fragment illustrates how to get the addresses of the `ncOpenObject`, `ncCloseObject`, and `ncRead` functions:

```
static NCTYPE_STATUS (_NCFUNC_ *PncOpenObject)
   (NCTYPE_STRING ObjName,
   NCTYPE_OBJH_P ObjHandlePtr);
static NCTYPE_STATUS (_NCFUNC_ *PncCloseObject)
   (NCTYPE_OBJH ObjHandle);
static NCTYPE_STATUS (_NCFUNC_ *PncRead)
   (NCTYPE_OBJH ObjHandle, NCTYPE_UINT32 DataSize,
   NCTYPE_ANY_P DataPtr);

PncOpenObject = (NCTYPE_STATUS (_NCFUNC_ *)
   (NCTYPE_STRING, NCTYPE_OBJH_P))
   GetProcAddress(NicanLib, (LPCSTR)"ncOpenObject");
PncCloseObject = (NCTYPE_STATUS (_NCFUNC_ *)

   (NCTYPE_OBJH))
   GetProcAddress(NicanLib, (LPCSTR)"ncCloseObject");
PncRead = (NCTYPE_STATUS (_NCFUNC_ *)
   (NCTYPE_OBJH, NCTYPE_UINT32, NCTYPE_ANY_P))
   GetProcAddress(NicanLib, (LPCSTR)"ncRead");
```

If `GetProcAddress` fails, it returns a NULL pointer. The following code fragment illustrates how to verify that none of the calls to `GetProcAddress` failed:

```
if ((PncOpenObject  == NULL) ||
       (PncCloseObject  == NULL) ||
       (PncRead  == NULL)) {
   FreeLibrary(NicanLib);
   printf("GetProcAddress failed");
}
```

Your application needs to de-reference the pointer to access an NI-CAN function, as illustrated by the following code:

```
NCTYPE_STATUS status;
NCTYPE_OBJH MyObjh;

status = (*PncOpenObject) ("CAN0", &MyObjh);
if (status < 0) {
   printf("ncOpenObject failed");
}
```

Before exiting your application, you need to free `nican.dll` with the following command:

```
FreeLibrary(NicanLib);
```

For more information on direct entry, refer to the Win32 SDK (Software Development Kit) online help.

Choosing Which NI-CAN Objects to Use

An application written for NI-CAN communicates on the network by using various objects. Which NI-CAN objects to use depends largely on the needs of your application. The following sections discuss the objects provided by NI-CAN, and reasons why you might use each class of object.

Using CAN Network Interface Objects

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

You use the CAN Network Interface Object to read and write complete CAN frames. As a CAN frame arrives from over the network, it can be placed into the read queue of the CAN Network Interface Object. You can retrieve CAN frames from this read queue using the `ncRead` function. For CAN Network Interface Objects, the `ncRead` function provides a timestamp of when the frame was received, the arbitration ID of the frame, the type of frame (data or remote), the data length, and the data bytes. You can also use the CAN Network Interface Object to write CAN frames using the `ncWrite` function.

Some possible uses for the CAN Network Interface Object include the following:

- You can use the read queue to log all CAN frames transferred across the network. This log is useful when you need to view preceding CAN traffic to verify that all CAN devices are functioning properly.

- You can use the write queue to transmit a sequence of CAN frames in quick succession. This is useful for applications in which you need to simulate a specific sequence of CAN frames to verify proper device operation.
- You can read and write CAN frames for access to configuration settings within a device. Because such settings generally are not accessed during normal device operation, a dedicated CAN Object is not appropriate.
- For higher level protocols based on CAN, you can use sequences of write/read transactions to initialize communication with a device. In these protocols, specific sequences of CAN frames often need to be exchanged before you can access the data from a device. In such cases, you can use the CAN Network Interface Object to set up communication, then use CAN Objects for actual data transfer with the device.

In general, you use CAN Network Interface Objects for situations in which you need to transfer arbitrary CAN frames.

Using CAN Objects

When a network frame is transmitted on a CAN based network, it always begins with what is called the arbitration ID. This arbitration ID is primarily used for collision resolution when more than one frame is transmitted simultaneously, but you can also use it as a simple mechanism to identify data. The CAN Object encapsulates a specific CAN arbitration ID and its associated data.

Every CAN Object is always associated with a specific CAN Network Interface Object, used to identify the physical interface on which the CAN Object is located. Your application can use multiple CAN Objects in conjunction with their associated CAN Network Interface Object.

The CAN Object provides high level access to a specific arbitration ID. You can configure each CAN Object for different forms of background access. For example, you can configure a CAN Object to transmit a data frame every 100 milliseconds, or to periodically poll for data by transmitting a remote frame and receiving the data frame response. The arbitration ID, direction of data transfer, data length, and when data transfer occurs (periodic or unsolicited) are all preconfigured for the CAN Object. When you have configured and opened the CAN Object, data transfer is handled in the background using read and write queues. For example, if the CAN Object periodically polls for data, the NI-CAN driver automatically handles the periodic transmission of remote frames, and stores incoming

data in the read queue of the CAN Object for later retrieval by the `ncRead` function. For CAN Objects that receive data frames, the `ncRead` function provides a timestamp of when the data frame arrived, and the data bytes of the frame. For CAN Objects that transmit data frames, the `ncWrite` function provides the outgoing data bytes.

Some possible uses for CAN Objects include the following:

- You can configure a CAN Object to periodically transmit a data frame for a specific arbitration ID. The CAN Object transmits the same data bytes repetitively until different data is provided using `ncWrite`. This configuration is useful for simulation of a device that periodically transmits its data, such as simulation of an automotive sensor. This configuration is also useful for devices that expect to periodically receive data for a particular arbitration ID to respond with data using a different arbitration ID, such as a device containing analog inputs and outputs.
- You can configure a CAN Object to watch for unsolicited data frames received for its arbitration ID, then store that data in the CAN Object's read queue. A watchdog timeout is provided to ensure that incoming data is received periodically. This configuration is useful when you want to apply a timeout to data received for a specific arbitration ID and store that data in a dedicated queue. If you do not need to apply a timeout for a given arbitration ID, it is often preferable to use the CAN Network Interface Object to receive that data.
- You can configure a CAN Object to periodically poll for data by transmitting a remote frame and receiving the data frame response. This configuration is useful for communication with devices that require a remote frame to transmit their data.
- You can configure a CAN Object to transmit a data frame whenever it receives a remote frame for its arbitration ID. You can use this configuration to simulate a device which responds to remote frames.

In general, you use CAN Objects for data transfer for a specific arbitration ID, especially when that data transfer needs to occur periodically.

Programming Model for NI-CAN Applications

The following steps demonstrate how to use the NI-CAN functions in your application. The steps are shown in Figure 2-1 in flowchart form.

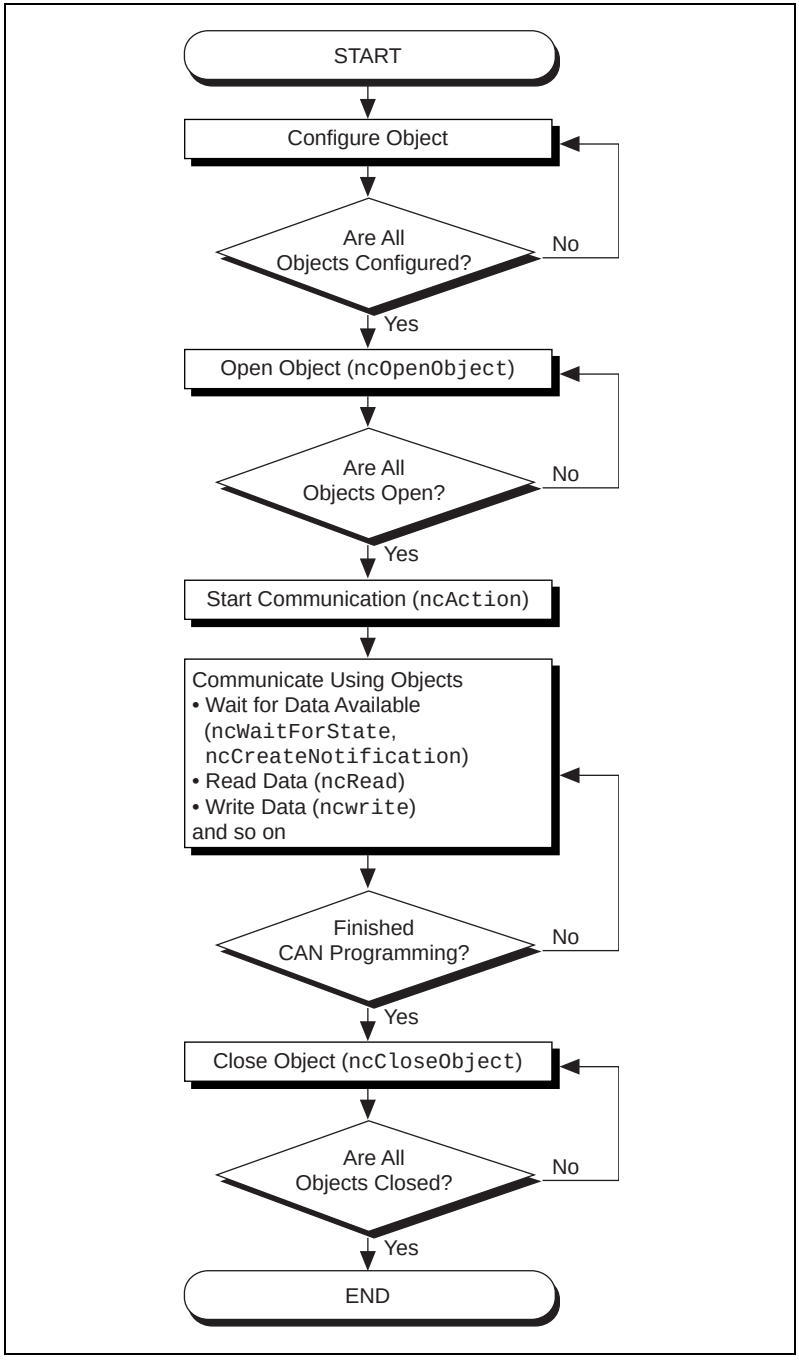

Figure 2-1. General Program Steps Using NI-CAN Functions

Step 1. Configure Objects

Prior to opening the objects used in your application, you must configure the objects with their initial attribute settings. You can configure the objects in one of two ways:

- You can use the NI-CAN Configuration utility to define your objects and specify their configuration attributes. This method is often preferable, because it does not require configuration to be handled within your application itself. Also, the NI-CAN Configuration utility provides online help that describes each of the configuration attributes.
- Each object can be configured within your application by calling the `ncConfig` function. This function takes the name of the object to configure, along with a list of configuration attribute settings.

Step 2. Open Objects

You must call the `ncOpenObject` function to open each object you use within your application.

The `ncOpenObject` function returns a handle for use in all subsequent NI-CAN calls for that object. When you are using the G language (LabVIEW) function library, this handle is passed through the upper left and right terminals of each NI-CAN function used after the open.

Step 3. Start Communication

You must start communication on the CAN network before you can use your objects to transfer data.

If you configured your CAN Network Interface Object to start on open, then that object and all of its higher level CAN Objects are started automatically by the `ncOpenObject` function, so nothing special is required for this step.

If you disabled the start-on-open attribute, then when your application is ready to start communication, use the CAN Network Interface Object to call the `ncAction` function with the `Opcode` parameter set to `NC_OP_START`. This call is often useful when you want to use `ncWrite` to place outgoing data in write queues prior to starting communication.

If you want to reset the CAN hardware completely to clear a pending Error Passive state, you can use the CAN Network Interface Object to call the `ncAction` function with the `Opcode` parameter set to `NC_OP_RESET`. This reset must be done prior to starting communication.

Step 4. Communicate Using Objects

After you open your objects and start communication, you are ready to transfer data on the CAN network. The manner in which data is transferred depends on the configuration of the objects you are using. For this example, assume that you are communicating with a CAN device that periodically transmits a data frame. To receive this data, assume that a CAN Object is configured to watch for data frames received for its arbitration ID and store that data in its read queue.

Step 4a. Wait for Available Data

To wait for the arrival of a data frame from the device, you can call `ncWaitForState` with the `DesiredState` parameter set to `NC_ST_READ_AVAIL`. The `NC_ST_READ_AVAIL` state tells you that data for the CAN Object has been received from the network and placed into the object's read queue. Another way to wait for the `NC_ST_READ_AVAIL` state is to call the `ncCreateNotification` function so you receive a callback when the state occurs. For more information on `ncWaitForState` and `ncCreateNotification`, refer to the *NI-CAN Programmer Reference Manual*.

When receiving data from the device, if your only requirement is to obtain the most recent data, you are not required to wait for the `NC_ST_READ_AVAIL` state. If this is the case, you can set the read queue length of the CAN Object to zero during configuration, so that it only holds the most recent data bytes. Then you can use the `ncRead` function as needed to obtain the most recent data bytes received.

Step 4b. Read Data

Read the data bytes using `ncRead`. For CAN Objects that receive data frames, `ncRead` returns a timestamp of when the data was received, followed by the actual data bytes (the number of which you configured in Step 1).

Steps 4a and 4b should be repeated for each data value you want to read from the CAN device.

Step 5. Close Objects

When you are finished accessing the CAN devices, close all objects using the `ncCloseObject` function before you exit your application.

Checking Status of Function Calls

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

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 2-2.

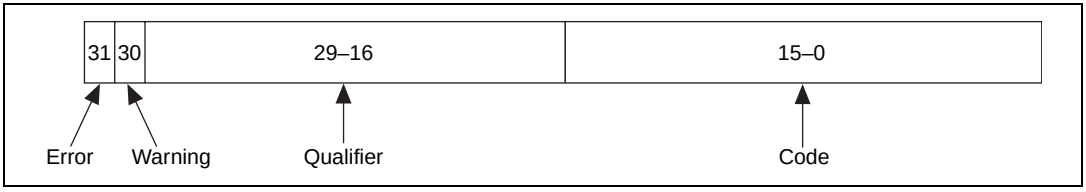

Figure 2-2. Status Format

Error and 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 has arisen which may require your attention. If no error or warning occurs, the entire status is set to zero to indicate success. Table 2-1 summarizes the behavior of NI-CAN status.

Table 2-1. 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 that 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).

For descriptions of the NI-CAN status codes and their qualifiers, refer to the *NI-CAN Programmer Reference Manual*.

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 NI-CAN's 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 in the *NI-CAN Programmer Reference Manual*. 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");
```

3

NI-CAN Programming Techniques

This chapter describes techniques for using the NI-CAN functions in your application.

For more detailed information about each NI-CAN function, refer to the *NI-CAN Programmer Reference Manual*.

Using Queues

To maintain an ordered history of data transfers, NI-CAN supports the use of queues, also known as FIFO (first-in-first-out) buffers. The basic behavior of such queues is common to all NI-CAN objects.

There are two basic types of NI-CAN queues: the read queue and the write queue. NI-CAN uses the read queue to store incoming network data items in the order they arrive. You access the read queue using `ncRead` to obtain the data. NI-CAN uses the write queue to transmit network frames one at a time using the network interface hardware. You access the write queue using `ncWrite` to store network data items for transmission.

State Transitions

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 and the queue is empty.

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.

Empty Queues

For both read and write queues, the behavior for reading an empty queue is similar. When you read an empty queue, the previous data item is returned again. For example, if you call `ncRead` when `NC_ST_READ_AVAIL` is false,

the data from the previous call to ncRead is returned 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. You should generally wait for NC_ST_READ_AVAIL prior to the first call to ncRead.

Full Queues

For both read and write queues, the behavior for writing a full queue is similar. When you write a full queue, NI-CAN returns the NC_ERR_OVERFLOW status codes. For example, if you write too many data items to a write queue, the ncWrite function eventually returns the overflow error.

Disabling Queues

If you do not need a complete history of all data items, you may prefer to disable the read queue and/or write queue by setting its length to zero. Using zero length queues generally saves memory, and often results in better performance. When a new data item arrives for a zero length queue, it overwrites the previous item without indicating an overflow. The NC_ST_READ_AVAIL and NC_ST_WRITE_SUCCESS states still behave as usual, but you can ignore them if you want only the most recent data. For example, when NI-CAN writes a new data item to the read buffer, the NC_ST_READ_AVAIL state becomes true until the data item is read. If you only want the most recent data, you can ignore the NC_ST_READ_AVAIL state, as well as the NC_ERR_OLD_DATA warning returned by ncRead.

Using the CAN Network Interface Object with CAN Objects

For many applications, it is desirable to use a CAN Network Interface Object in conjunction with higher level CAN Objects. For example, many CAN devices require a specific sequence of CAN frames to initialize for data transfer. For such devices, you can use the CAN Network Interface Object to transmit and receive frames required for initialization, then use CAN Objects for data transfer (such as transmitting a periodic request for data). For more information on the different uses of NI-CAN objects, refer to the *Choosing Which NI-CAN Objects to Use* section in Chapter 2, *Developing Your Application.*

When one or more CAN Objects are open, the CAN Network Interface Object cannot receive frames which would normally be handled by the CAN Objects. The flowchart in Figure 3-1 shows the steps performed by NI-CAN when a CAN frame is received.

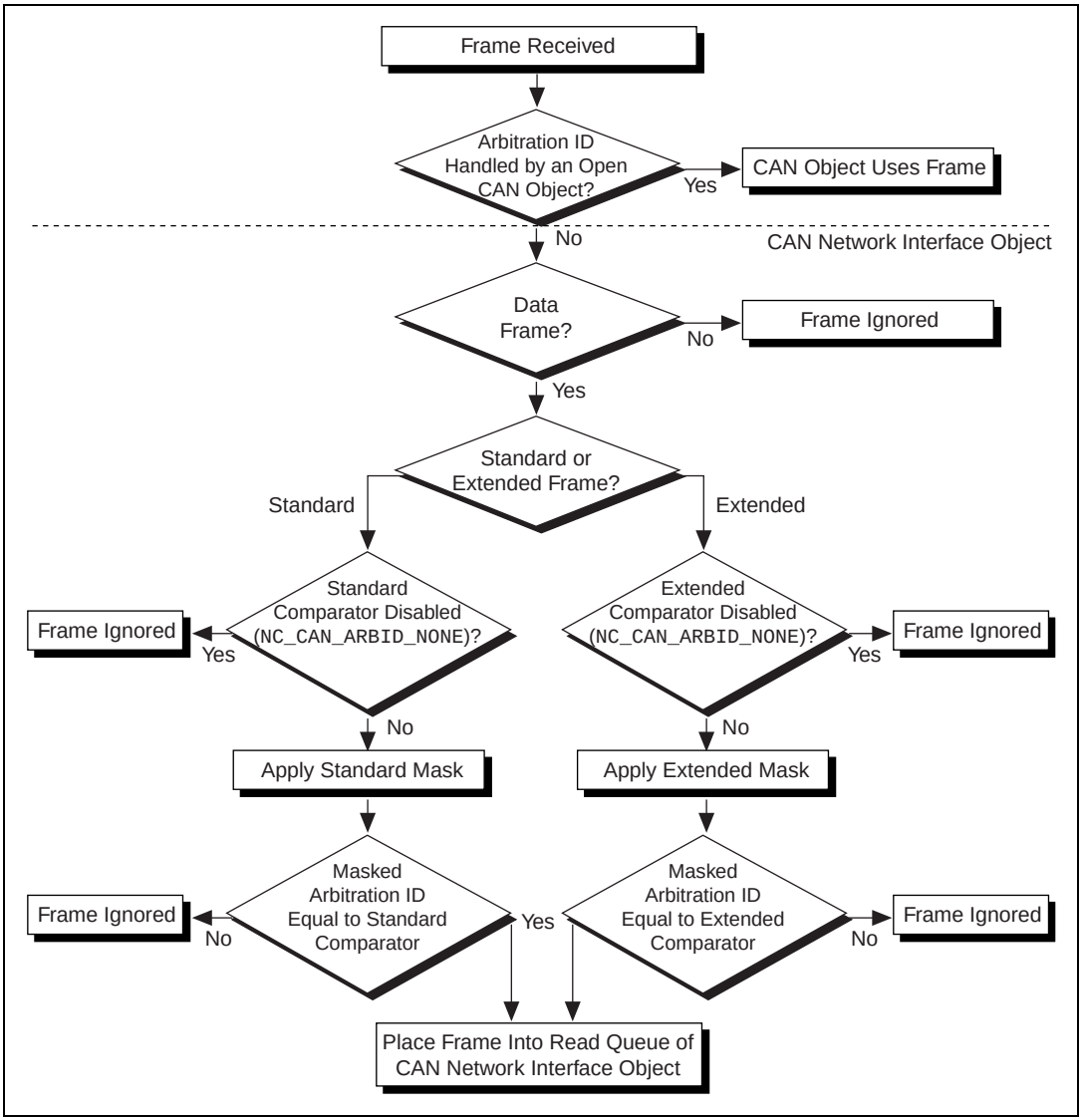

Figure 3-1. Flowchart for CAN Frame Reception

The decisions in Figure 3-1 are generally performed by the on-board CAN communications controller chip. Nevertheless, if you intend to use CAN Objects as the sole means of receiving CAN frames, it is best to disable all frame reception in the CAN Network Interface Object by setting the comparator attributes to NC_CAN_ARBID_NONE. By doing this, the CAN communications controller chip is best able to filter out all incoming frames except those handled by CAN Objects.

Detecting State Changes

You can detect state changes for an object 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.

Use the ncGetAttribute function when you need to determine the current state of an object. For example, if you want to determine whether a background error has occurred for an object, you can use ncGetAttribute to obtain the state and check for NC_ST_ERROR.

Use the ncWaitForState function when your application must wait for a specific state before proceeding. For example, if you call ncWrite to write a frame, and your application cannot proceed until the frame is successfully transmitted, you can call ncWaitForState to wait for NC_ST_WRITE_SUCCESS.

Use the ncCreateNotification function when your application must handle a specific state, but can perform other processing while waiting for that state to occur. The ncCreateNotification function registers a callback function, which is invoked when the desired state occurs. For example, a callback function for NC_ST_READ_AVAIL can call ncRead and place the resulting data in a buffer. Your application can then perform any tasks desired, and process the CAN data only as needed.

4

Application Examples

This chapter describes the sample applications provided with your NI-CAN software.

The examples in this chapter are designed to illustrate basic NI-CAN programming, as well as specific concepts and techniques that can help you write your own applications. The description of each example includes the programmer's task, a program flowchart, and numbered steps that correspond to the numbered blocks on the flowchart.

The following example programs are included with your NI-CAN software:

- `obj2obj.c` is the C source code file for Example 1. `obj2obj.vi` is the LabVIEW source code file for Example 1. In this example, one CAN Object is used to periodically transmit data to another CAN Object.
- `simpanlz.c` is the C source file for Example 2. This example illustrates a simple CAN bus analyzer using the CAN Network Interface Object. It also demonstrates usage of the `ncCreateNotification` function.
- `interact.vi` is the LabVIEW source code file for Example 3. In this example, one CAN Network Interface Object and one CAN Object are used to transmit and receive CAN frames interactively.
- `ReadMult.c` is the C source file for the `ncReadMult` function. This example shows how to use the function to retrieve CAN frames from the read queue. The example can be used as the basis for building analyzer-style applications to retrieve high-speed incoming data for future analysis. When used with the Network Interface Object, this function allows more data processing time because it retrieves multiple frames in one call, as opposed to the `ncRead` function that retrieves one frame per call.
- `ReadMultNet.vi` and `ReadMultObj.vi` are LabVIEW/BridgeVIEW examples for using the `ncReadNetMult.vi` and `ncReadObjMult.vi`. The primary difference between the two vis is the data cluster used to output the received data. The `ncReadNetMult.vi` use the `NCTYPE_CAN_FRAME_TIMED` cluster

and ncReadObjMult.vi uses the NCTYPE_CAN_DATA_TIMED cluster.

- SimpanlzLS.c is the C source that shows the use of the NC_ATTR_LOG_COMM_ERRS attribute for low-speed CAN applications.
- InteractLS.vi is the LabVIEW/BridgeVIEW example that shows the ncNetAttrLS cluster that lets you set the NC_ATTR_LOG_COMM_ERRS attribute for low-speed CAN applications

Example 1. Using CAN Objects

This example focuses on the basics of using CAN Objects.

An automotive test engineer is trying to test a variety of CAN devices. One of the CAN devices is a speed display. This display expects to receive the current speed of the vehicle in a CAN frame every 100 milliseconds, so that the driver of the vehicle can be continuously updated. Another CAN device is a speed sensor (speedometer), which measures the speed of the vehicle and transmits it in a CAN frame every 100 milliseconds.

To use NI-CAN to test the speed display, the engineer uses a CAN Object to simulate the role of the speed sensor. This CAN Object is configured to transmit a simulated speed every 100 milliseconds. By using the CAN Object to transmit different speeds, the test engineer can verify that the speed display always shows the correct speed.

To use NI-CAN to test the speed sensor, the engineer uses a CAN Object to simulate the role of the speed display. This CAN Object is configured to receive speeds from the sensor and display them. By using this CAN Object to receive and display different speeds, the test engineer can connect the speed sensor to a real engine, then verify that the speeds it transmits are correct.

To learn the basics of CAN Object usage prior to testing the actual devices, the test engineer writes a simple example to implement both CAN Objects. To do this, he uses a two-port CAN interface, such as the AT-CAN/2. He connects the top port of the card to the bottom port using a cable. One port plays the role of the simulated speed display, and the other port plays the role of the simulated speed sensor.

Example 1 configures one CAN Object to receive data, and configures another CAN Object to transmit data. Both CAN Objects use arbitration ID 5. The data is transmitted every second, so the test engineer can view each period's data as well as its timestamp. Once the engineer completes the example, he can change it for testing of each device by using one CAN Object at a time.

The following steps correspond to the program flow chart in Figure 4-1.

1. The application calls `ncConfig` to configure the CAN Network Interface Objects for both ports (`CAN0` and `CAN1`). Normally, this configuration would be handled using the NI-CAN Configuration utility, but `ncConfig` is used instead to keep the example self-contained.
2. The application calls `ncConfig` to configure the CAN Objects for both ports (`CAN0::STD5` and `CAN1::STD5`). Once again, such configuration would normally be handled using the NI-CAN Configuration utility.
3. The application calls `ncOpenObject` to open the two CAN Objects.
4. The application calls `ncWrite` for `CAN1::STD5`. This call starts the periodic transmission of data. For this example, the same data is transmitted every period.
5. The application calls `ncWaitForState` for `CAN1::STD5` in order to wait for the `NC_ST_WRITE_SUCCESS` state. This state is set when the first CAN frame is successfully transmitted to the other CAN Object.
6. The application calls `ncWaitForState` for `CAN0::STD5` in order to wait for the `NC_ST_READ_AVAIL` state. This state is set when a CAN frame is received from the other CAN Object.
7. The application calls `ncRead` to read data for `CAN0::STD5`. The data contains the value written in step 4, as well as a timestamp of when the value arrived.
8. The application loops back to step 6 for a total of 10 periods. Each period, step 6 waits one second before the next data value is received.
9. When all 10 loops complete, both CAN Objects are closed using `ncCloseObject`.

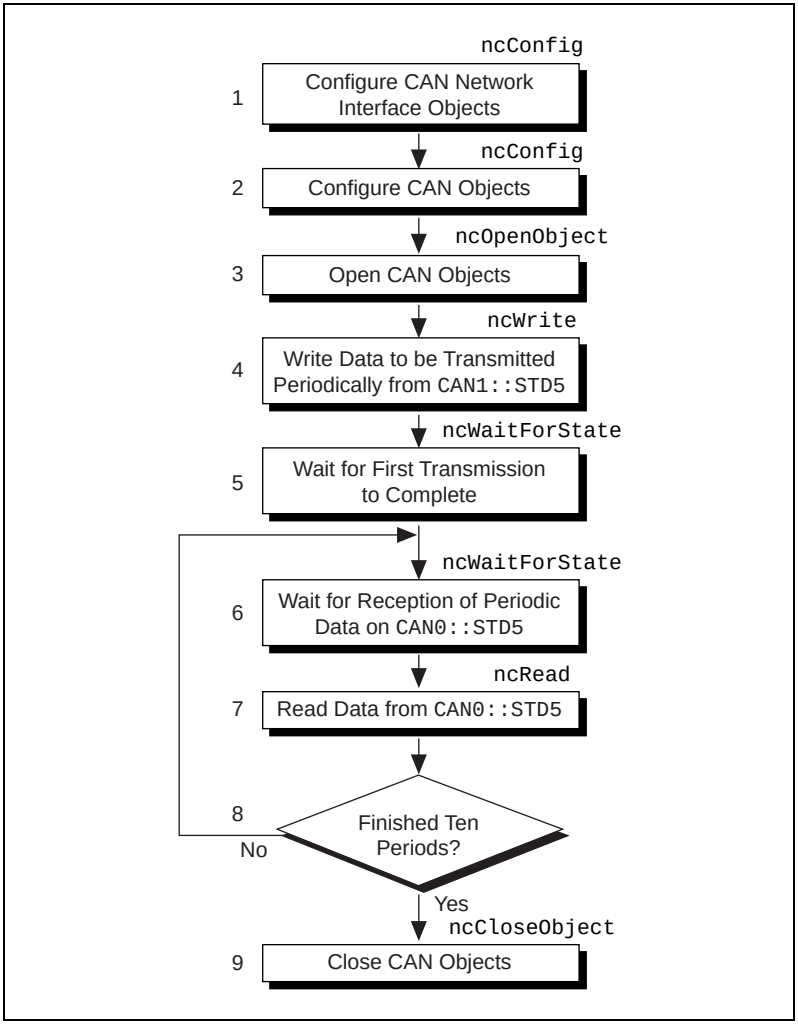

Figure 4-1. Program Flowchart for Example 1

Example 2. Simple CAN Bus Analyzer

This example focuses on usage of the `ncCreateNotification` function within the C programming language. It illustrates a simple CAN bus analyzer using the CAN Network Interface Object.

An automotive test engineer is writing a diagnostic utility for use in service bays. This utility is used to diagnose problems with car models that use

CAN as their in-vehicle network. The utility monitors the car's CAN network traffic to test for defective devices, incorrect sensor data, and so on.

In developing the utility, the test engineer wants two threads. One thread receives data from the CAN network and places it into a buffer. The other thread processes the data in the buffer in order to check for erroneous network traffic.

The following steps correspond to the program flowchart in Figure 4-2.

1. The application calls `ncConfig` to configure the CAN Network Interface Object used for bus analysis (`CAN0`). The masks and comparators are configured such that all CAN data frames are received. Normally, this configuration would be handled using the NI-CAN Configuration utility, but `ncConfig` is used instead to keep the example self-contained.
2. The application calls `ncOpenObject` to open the CAN Network Interface Object.
3. The application calls `ncCreateNotification` to create the notification thread, which is used to receive frames into the buffer. This is done by registering a callback function for the `NC_ST_READ_AVAIL` state. After creating the notification thread, the main thread proceeds to Step 7.
4. The notification thread remains idle until its callback function is invoked by the NI-CAN driver.
5. If the callback function detects the `NC_ST_READ_AVAIL` state, `ncRead` is called to read the frame, and the frame is placed into the buffer for processing by the main thread.
6. If the callback function detects a timeout or error, it indicates the problem to the main thread, then proceeds to Step 9. If no timeout or error is detected, the callback function re-enables the notification and returns to Step 4 (idle).
7. If the main thread detects a new frame in the buffer (placed there by the notification thread in Step 5), it processes the frame. For this example, processing the frame merely entails printing it to the screen.
8. If the main thread does not detect a timeout or error, it loops back to Step 7 to wait for more frames.
9. If a timeout or error occurs, the main thread prints it to the screen. The timeout occurs if no frame is received within 30 seconds.
10. The application calls `ncCloseObject` to close the CAN Network Interface Object.

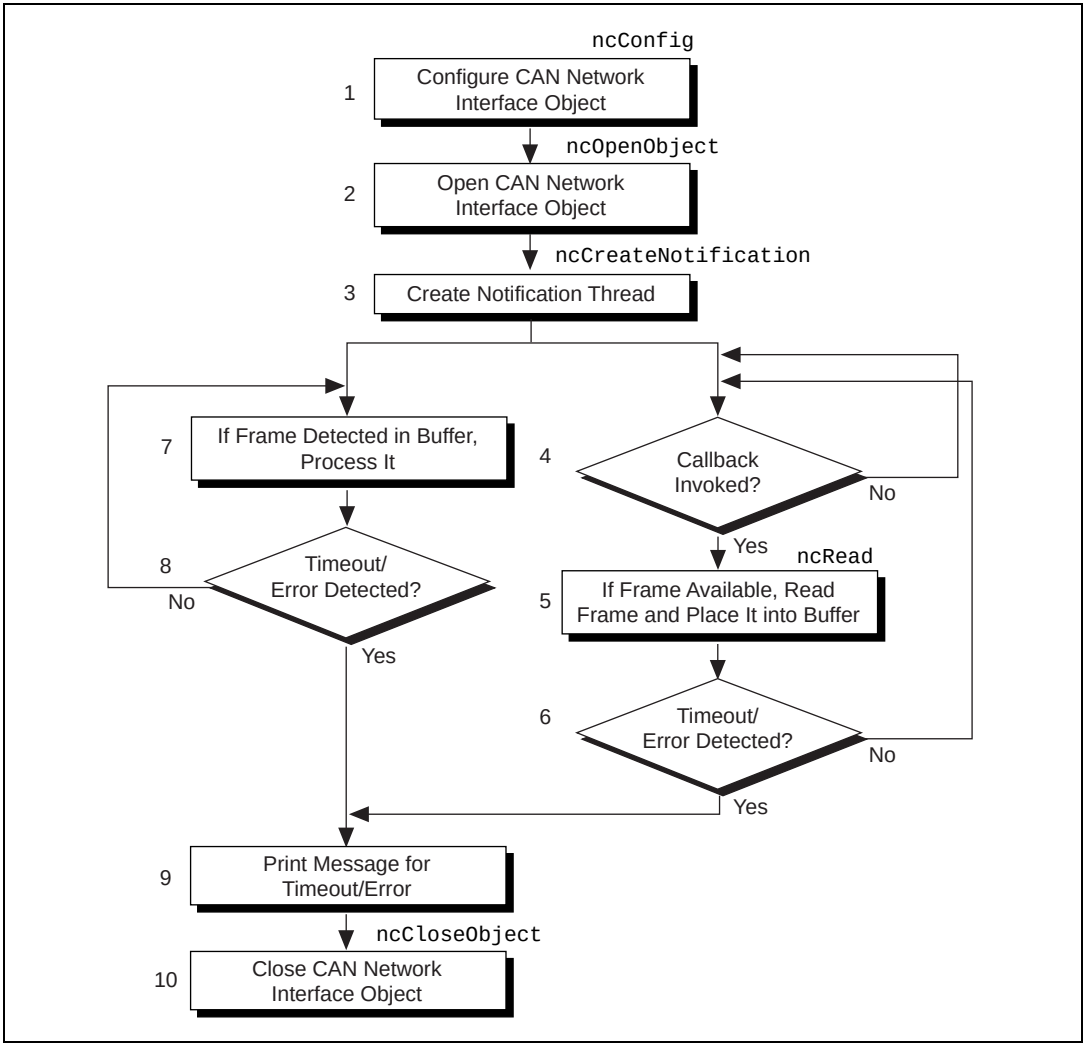

Figure 4-2. Program Flowchart for Example 2

Example 3. Interactive CAN Example

This example provides an overview of how the features of NI-CAN are used within LabVIEW. It provides a LabVIEW front panel that you can use to interact with CAN devices and to learn the basic operation of NI-CAN.

The `interact.vi` front panel provides **CAN Network Interface** and **Baud Rate** controls, which are used to specify the CAN interface to use

(such as CAN0), as well as the communication baud rate (such as 125000). All frames received are displayed in an array of **Received Frames**. If you want to transmit a specific frame, you can enter the desired Arbitration Id, Is Remote flag (off means CAN data frame), Data Length, and Data bytes, then select WRITE to transmit.

The interact.vi front panel also supports an optional CAN Object configured as Transmit Data Periodically. Before running the example, select **Configure Periodic Transmit** to use this object, and also select the Arbitration Id, Data Length, and Period to configure. While the example is running, you can use the **Periodic Transmit Data** control to update the data transmitted each period.

For more information on how to use the front panel of interact.vi, scroll up to the help text located above the front panel controls.

If you do not have a CAN device with which to experiment using interact.vi, but you have a two-port CAN interface (such as the PCI-CAN/2), you can use two copies of interact.vi to experiment. Save a separate copy of the example (such as interact2.vi), then run one copy on one port (such as CAN0) and the other copy on the other port (such as CAN1).

The following steps correspond to the program flowchart in Figure 4-3.

1. The application calls ncConfig to configure the CAN Network Interface Object. The name of the object and its baud rate are obtained from front panel controls.
2. The application calls ncOpenObject to open the CAN Network Interface Object.
3. If **Configure Periodic Transmit** is checked, the application calls ncConfig to configure the CAN Object. The name of the object is obtained using the front panel arbitration ID. The data length and period are also obtained from front panel controls.
4. If **Configure Periodic Transmit** is checked, the application calls ncOpenObject to open the CAN Object.
5. If the **WRITE** button has been selected, the front panel arbitration ID, remote/data flag, data length, and data are used to call ncWrite for the CAN Network Interface Object.
6. The ncRead function is called for the CAN Network Interface Object, to see if a CAN frame has been received. If ncRead returns a code of zero (success) in its error out cluster, the received CAN frame is inserted into the **Received Frames** array.

7. If **Configure Periodic Transmit** is checked, the application calls `ncWrite` for the CAN Object to update the data used for periodic transmissions.
8. If no NI-CAN error has occurred and the **Stop** button has not been selected, the application loops back to Step 5.
9. The application calls `ncCloseObject` to close the CAN Network Interface Object, then calls `ncCloseObject` to close the CAN Object.

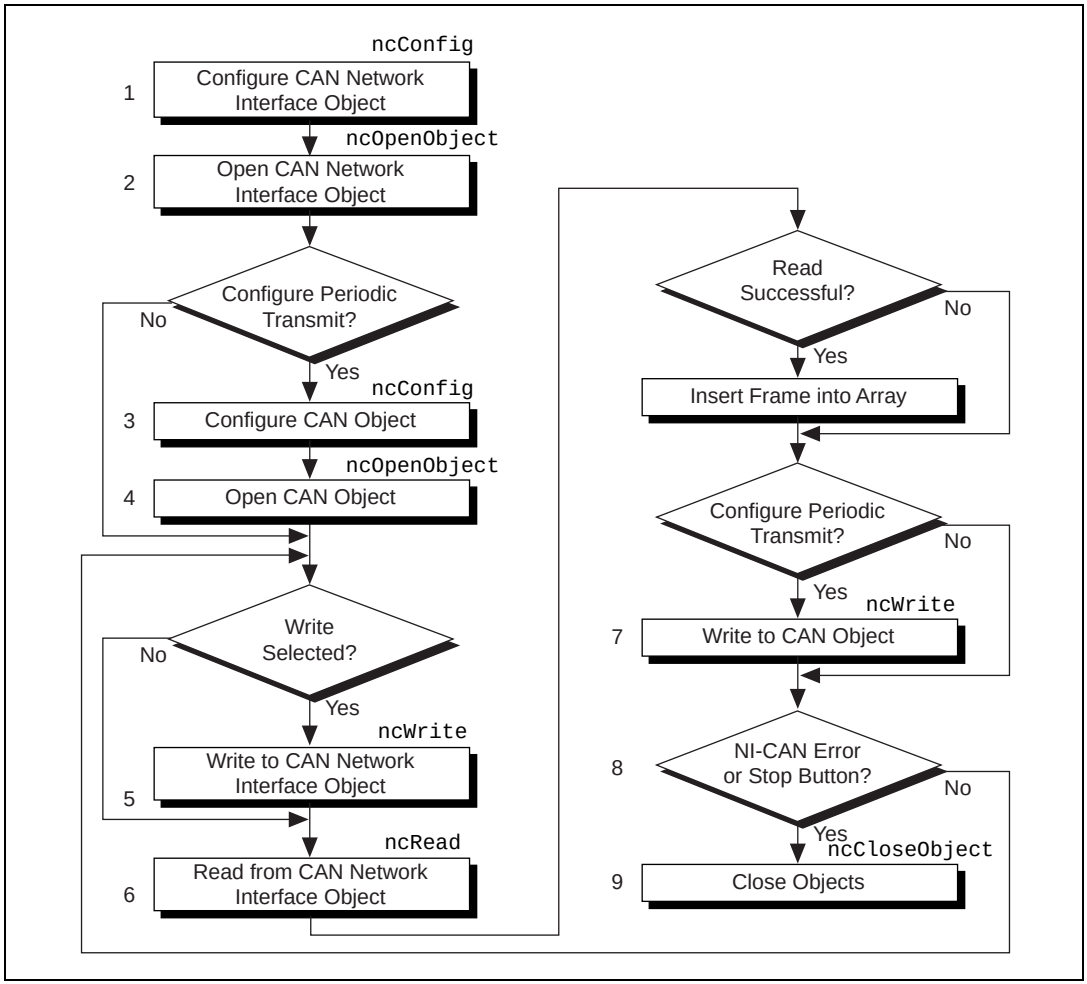

Figure 4-3. Program Flowchart for Example 3

5

NI-CAN Configuration Utility

This chapter describes the NI-CAN Configuration utility you can use to configure the objects of the NI-CAN software.

Overview

The Windows 98/95 NI-CAN Configuration utility is integrated into the Windows Device Manager. The Windows NT NI-CAN Configuration utility is integrated into the Windows NT Control Panel. You can use the NI-CAN Configuration utility to view or modify the configuration of NI-CAN objects. For each CAN interface in your system, you can use the NI-CAN Configuration utility to configure each CAN port as a CAN Network Interface Object. For example, you can configure the two ports of an AT-CAN/2 as `CAN0` and `CAN1`, and you can initialize configuration attributes such as baud rate. For each CAN Network Interface Object, you can use the NI-CAN Configuration utility to create and modify CAN Objects. The online help includes all of the information that you need to properly configure the objects of the NI-CAN software.

The NI-CAN Configuration utility provides an easy mechanism for configuring the objects used by your application. By configuring objects with the NI-CAN Configuration utility, your application can open the objects and begin using them. If you do not want your application to rely on the NI-CAN Configuration utility, it must call the `ncConfig` function for every object you use. The `ncConfig` function specifies values for all of an object's configuration attributes. The configuration attributes you specify using `ncConfig` override any configuration attributes you have specified using the NI-CAN Configuration utility. For more information on `ncConfig`, refer to the *NI-CAN Programmer Reference Manual.*

Starting the NI-CAN Configuration Utility in Windows 98/95

To start the NI-CAN Configuration utility on Windows 98/95, follow these steps.

1. Double-click on the **System** icon in the **Control Panel**, which can be opened from the **Settings** selection of the **Start** menu.
2. Select the **Device Manager** tab in the **System Properties** dialog box that appears.
3. Click on the **View devices by type** button at the top of the **Device Manager** tab, and double-click on **National Instruments CAN Interfaces**.
4. In the list of installed interfaces immediately below **National Instruments CAN Interfaces**, double-click on the particular interface type you want to configure. If you have only one National Instruments CAN interface in your computer, only one interface type appears in the list. If an exclamation point or an X appears next to the interface, there is a problem, and you should refer to the *Problem Shown in Device Manager* section of Appendix B, *Windows 98/95: Troubleshooting and Common Questions*, to resolve your problem before you continue. Use the **Resources** tab to provide information about the hardware resources assigned to the CAN interface, and use the **Settings** tab to configure the objects for the CAN interface. For information on using the **Settings** tab to configure your objects, refer to *Configuring Objects with the NI-CAN Configuration Utility*, later in this chapter.

Starting the NI-CAN Configuration Utility in Windows NT

To start the NI-CAN Configuration utility on Windows NT, open your Windows NT Control Panel, and double-click on **NI-CAN Configuration**.

Because you can use the NI-CAN Configuration utility to modify the configuration of the NI-CAN kernel drivers, you must be logged on to Windows NT as the `Administrator` to make any changes. If you start the NI-CAN Configuration utility without `Administrator` privileges, it runs in read-only mode; you can view the settings, but you cannot make changes.

The main dialog box of the NI-CAN Configuration utility for Windows NT contains a list of all National Instruments CAN interfaces in your computer. For each CAN interface, the **Resources** button opens a dialog box you can use to specify hardware resources for the CAN interface, and the **Settings** button opens a dialog box you can use to configure the objects for the CAN interface. For information on using the **Settings** tab to configure your objects, refer to *Configuring Objects with the NI-CAN Configuration Utility*, later in this chapter.

Note Because the current version of Windows NT is not fully Plug and Play, you must specify valid hardware resources for the CAN interface using the Resources button before using your National Instruments CAN interface with Windows NT. For information on verifying proper resource assignment, refer to your NI-CAN getting started manual.

After you have finished configuring your CAN interfaces, click on the **OK** button to close the dialog box.

Configuring Objects with the NI-CAN Configuration Utility

Figure 5-1 shows the **Settings** dialog box for an AT-CAN/2. The dialog box shown is for Windows 98/95, but the Windows NT dialog box is similar.

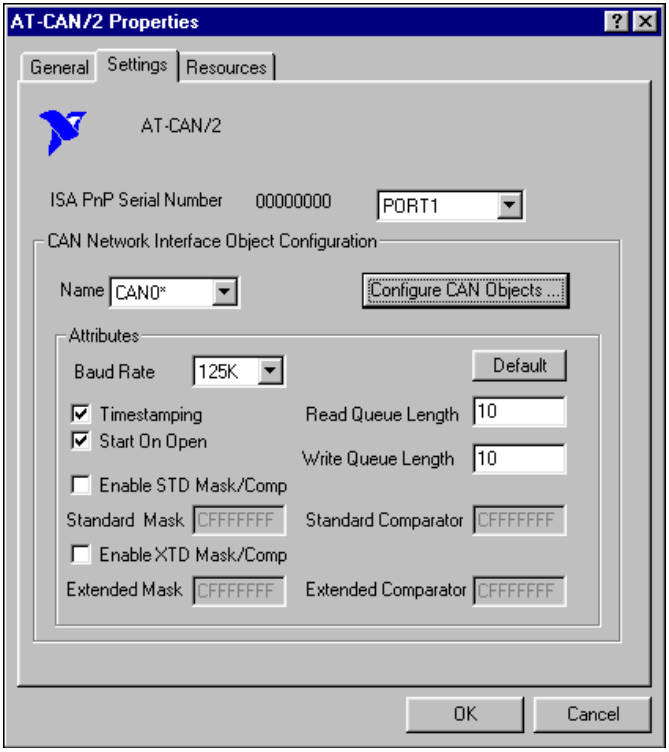

Figure 5-1. NI-CAN Settings Dialog Box for an AT-CAN/2

Hardware information appears at the top of the dialog box, so you can differentiate the selected CAN interface from others of the same type. For example, the **ISA PnP Serial Number** is provided for the AT-CAN. This serial number is printed physically on the interface, and you can use it to distinguish multiple AT-CAN interfaces installed in the same computer.

To access online help for the NI-CAN Configuration utility, right-click the mouse anywhere on the **Settings** tab, and select **Full Help** from the pop-up menu that appears. Alternately, you can select **What's This?** from the pop-up menu to see context-sensitive help for the item you have clicked on.

Select the Port

For two-port CAN interfaces such as the AT-CAN/2, a list box at the top of the **Settings** tab allows you to select which port to configure. This port number is printed physically next to the CAN connectors on the back of the interface, with **Port 1** as the top port, and **Port 2** as the bottom port. For one-port CAN interfaces such as the AT-CAN, the list box always lists **Port 1**.

Select the CAN Network Interface Object Name

After you have selected the port to configure, use the **Name** drop-down box to select the name of the CAN Network Interface Object (`CAN0`, `CAN1`, and so on). Your application uses the CAN Network Interface Object name as a logical reference to the port. You must assign a CAN Network Interface Object name for each port of every National Instruments CAN interface in your computer.

In the **Name** drop-down box, a small X appears after each name that has already been assigned to a physical port. This indication should assist you in assigning a unique name to each port in your system.

Specify the Configuration Attributes

Use the controls in the **Attributes** section to specify the configuration attributes for the CAN Network Interface Object. The attribute settings are associated with the physical port, and thus remain the same even if you decide to change the **Name** of the port. A control is provided for each configuration attribute of the CAN Network Interface Object. Within the **Attributes** section, you can use the **Default** button to initialize the attribute controls with acceptable default values.

If you need help with a particular control, click on the question mark near the upper right corner of the dialog box, then click on the control in question to view the context-sensitive online help. Alternately, you can right-click the mouse on the control, and then select **What's This?** from the pop-up menu that appears.

Configure the CAN Objects

Selecting the **Configure CAN Objects** button opens a dialog box you can use to configure the CAN Objects for the selected port. Figure 5-2 shows the **CAN Object Configuration** dialog box for Windows 98/95. The Windows NT dialog box is similar.

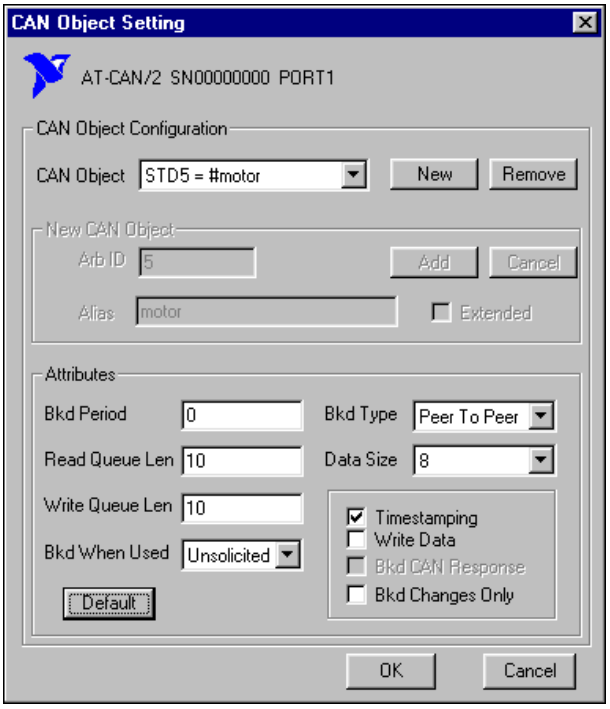

Figure 5-2. CAN Object Configuration Dialog Box

If you need help on a particular control in this dialog box, right-click the mouse on the control, and then select **What's This?** from the pop-up menu that appears.

Select the CAN Object

Use the **CAN Object** list box to select the CAN Object you want to configure. The list box lists all existing CAN Objects. The names are the same as those used with NI-CAN, with the arbitration ID of the CAN Object shown in decimal format (`STD5`, `XTD12004`, and so on). If an alias has been assigned for a CAN Object, the alias is listed after the name of the NI-CAN Object.

Add CAN Object Configurations

When you want to add a new CAN Object configuration, click on the **New** button to enable the controls in the **New CAN Object** section.

Use the **Arb ID** control to enter the decimal arbitration ID for the new CAN Object. Use the **Extended** check box to specify whether the arbitration ID is standard (unchecked) or extended (checked).

You can use the optional **Alias** control in the **New CAN Object** section to enter a user-defined alias for the CAN Object. You can use this alias with calls to `ncOpenObject` as a substitute for the complete object name. For example, if you add a CAN Object with arbitration ID 5 to the port named `CAN0`, then enter an alias `MotorSpeed`, you can open the name `#MotorSpeed` instead of the complete name `CAN0::STD5`. For more information on user-defined aliases, refer to the description of `ncOpenObject` in the *NI-CAN Programmer Reference Manual*.

After entering the arbitration ID, click on the **Add** button to add the CAN Object to the list. You may then specify the attributes for the CAN Object. To cancel the addition of the new CAN Object, click on the **Cancel** button.

Remove CAN Object Configurations

If you want to remove a CAN Object configuration, select the object from the **CAN Objects** list, then click on the **Remove** button.

Specify the Configuration Attributes

After adding a new CAN Object or selecting an existing CAN Object, you can use the controls in the **Attributes** section to specify the configuration attributes for the CAN Object.

Within the **Attributes** section, you can use the **Default** button to initialize the attribute controls with acceptable default values.

If you need help with a particular control, click on the question mark near the upper right corner of the dialog box, then click on the control in question to view the context-sensitive online help. Alternately, you can right-click the mouse on the control, and then select **What's This?** from the pop-up menu that appears.

Exit the CAN Object Setting Dialog Box

When you are finished configuring the CAN Objects, click on **OK** to close the dialog box and save any changes you have made, or click **Cancel** to close the dialog box without saving any changes you have made.

After selecting **OK** or **Cancel**, you are returned to the **Settings** tab so you can complete the configuration for the CAN interface.

Complete the Configuration

When you have finished using the NI-CAN Configuration utility, click **OK** to close the dialog box and saves any changes you have made, or click **Cancel** to close the dialog box without saving any changes you have made.

A

Uninstalling the Hardware and Software

This appendix describes how to uninstall the CAN hardware and the NI-CAN software.

Uninstalling the CAN Hardware from Windows 98/95

Before physically removing the CAN hardware from the computer, you must remove the hardware information from the Windows Device Manager.

To remove the hardware information from Windows 98/95, double-click on the **System** icon in the **Control Panel**, which you can open from the **Settings** selection of the **Start** menu. Select the **Device Manager** tab in the **System Properties** dialog box that appears, click the **View devices by type** button at the top of the **Device Manager** tab, and double-click on the **National Instruments CAN Interfaces** icon.

To remove an interface, select it from the list of interfaces under **National Instruments CAN Interfaces** as shown in Figure A-1, and click the **Remove** button.

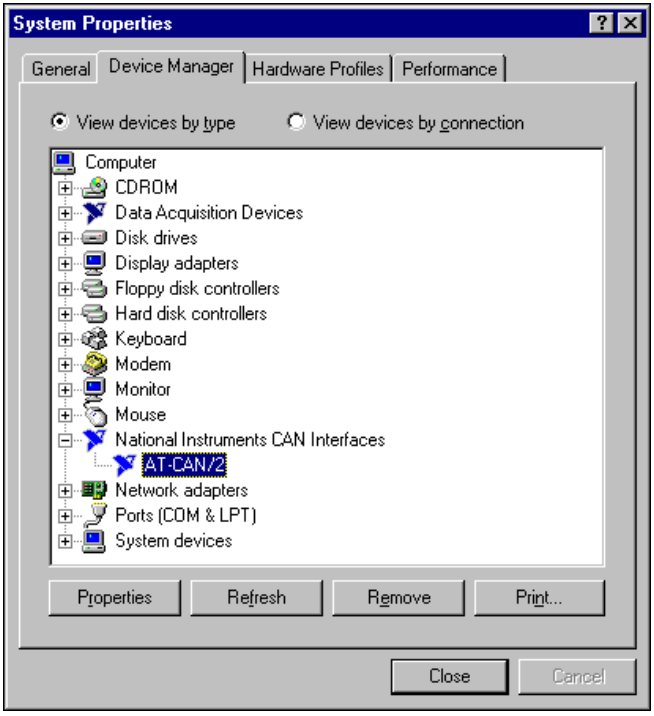

Figure A-1. Selecting an Interface to Remove from Windows 98/95

After you remove the appropriate CAN interface information from the Device Manager, you should shut down Windows, power off your computer, and physically remove the CAN interfaces.

Uninstalling the CAN Hardware from Windows NT

Because the current version of Windows NT does not maintain hardware information for the CAN interfaces, you need only to physically remove the CAN interfaces from your computer. Power off your computer and physically remove the CAN interfaces.

Uninstalling the NI-CAN Software

Before uninstalling the NI-CAN software, you should remove all CAN interface hardware from your computer.

Complete the following steps to remove the NI-CAN software.

1. Run the **Add/Remove Programs** applet from the **Control Panel**. A dialog box similar to the one in Figure A-2 appears. This dialog box lists the software available for removal.

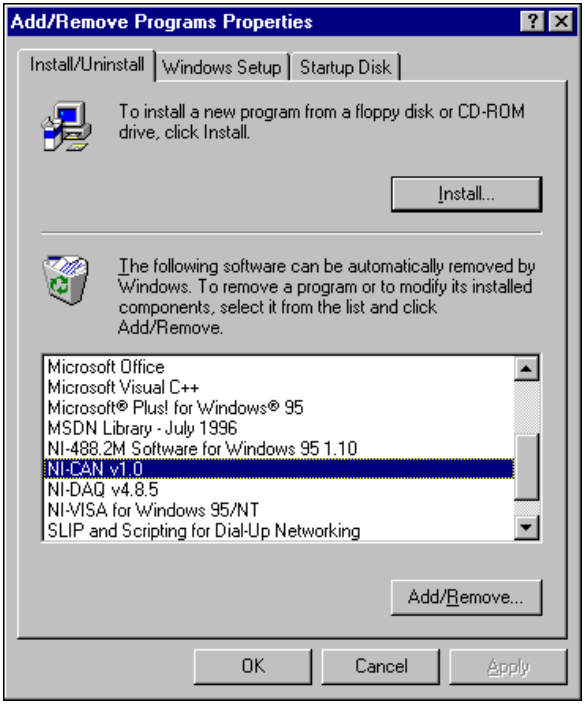

Figure A-2. Add/Remove Programs Properties Dialog Box

2. Select the NI-CAN software you want to remove, and click the **Add/Remove** button. The uninstall program runs and removes all folders, utilities, device drivers, DLLs, and registry entries associated with the NI-CAN software. Figure A-3 shows the results of a successful uninstallation.

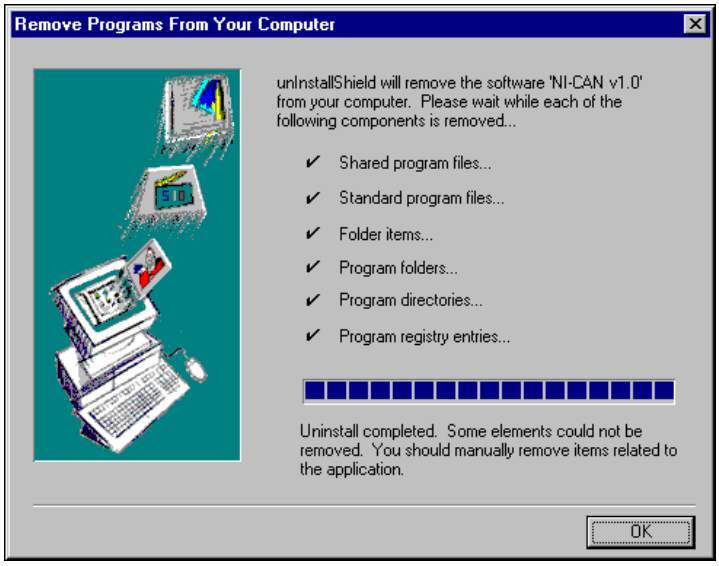

Figure A-3. NI-CAN Uninstallation Results

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory, because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

After the uninstall program completes, select **OK**, then restart your computer.

If you want to reinstall the hardware and software, refer to the getting started manual.

B

Windows 98/95: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems with the NI-CAN software for Windows 98/95 and answers some common questions.

Troubleshooting Windows Device Manager Problems

The Windows Device Manager contains configuration information for all of the CAN hardware it is aware of that is installed in your system. To start the Windows Device Manager, double-click on the **System** icon under **Start»Settings»Control Panel**. In the **System Properties** box that appears, select the **Device Manager** tab and click the **View devices by type** radio button at the top of the tab.

If there is no **National Instruments CAN Interfaces** item and you are certain you have a CAN interface installed, refer to the *No National Instruments CAN Interfaces* section of this appendix.

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, refer to the *Missing CAN Interface* section of this appendix.

If the CAN interface you are looking for is listed, but has a circled X or exclamation mark (!) over its icon, refer to the *Problem Shown in Device Manager* section of this appendix.

No National Instruments CAN Interfaces

If you are certain you have a Plug and Play CAN interface installed, but no **National Instruments CAN Interfaces** item appears in the **Device Manager**, the interface is probably incorrectly listed under **Other Devices**. Double-click on the **Other Devices** item in the Device Manager and, one by one, remove each National Instruments CAN interface listed there by selecting its name and then clicking the **Remove** button. After all of the

National Instruments CAN interfaces have been removed from **Other Devices**, click the **Refresh** button. At this point, the system rescans the installed hardware, and the CAN interface should show up under **National Instruments CAN Interfaces** without any problems. If the problem persists, contact National Instruments.

Missing CAN Interface

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, the CAN interface is not properly installed. For National Instruments CAN hardware, this problem indicates that the interface is not physically present in the system.

Problem Shown in Device Manager

If a CAN interface is not working properly, its icon has a circled X or exclamation mark (!) overlaid on it, as shown in Figure B-1.

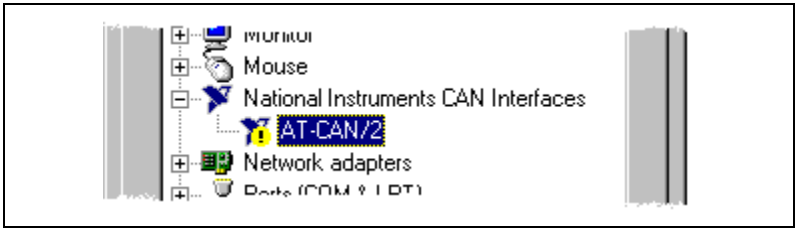

Figure B-1. CAN Interface That Is Not Working Properly

This problem can occur for several reasons. If you encounter this problem, the Device Manager should list an error code that indicates why the problem occurred. To see the error code for a particular interface, select the name of the interface and click on the **Properties** button to go to the **General** tab for that CAN interface. The **Device Status** section of the **General** tab shows the error code. Locate the error code in the following list to find out why your CAN interface is not working properly:

- Code 8—The NI-CAN software was incompletely installed. To solve this problem, reinstall the NI-CAN software.
- Code 9—Windows 98/95 had a problem reading information from the CAN interface. Contact National Instruments for assistance.
- Code 12—The CAN interface was not assigned a physical memory range. If your computer does not have 8 KB of available memory, Windows 98/95 might configure your CAN interface without a physical memory assignment. The NI-CAN software cannot function without 8 KB of physical memory. Another way to verify this problem

is to look at the **Resource settings** list on the **Resources** tab to verify that the CAN interface was not assigned a Memory Range. To solve this problem, free up an 8 KB Memory Range (such as D0000 to D1FFF hex) that is being used by another device in the system.

- Code 15—The CAN interface was not assigned an Interrupt Request level. If your computer does not have any available Interrupt Request levels, Windows 98/95 might configure your CAN interface without an Interrupt Request level. The NI-CAN software cannot function without an Interrupt Request level. Another way to verify this problem is to look at the **Resource settings** list on the **Resources** tab to verify that the CAN interface was not assigned an Interrupt Request level. To solve this problem, free up an Interrupt Request level that is being used by another device in the system.
- Code 22—The CAN interface is disabled. To enable the CAN interface, check the appropriate configuration checkbox in the **Device Usage** section of the **General** tab.
- Code 24—The CAN interface is not present, or the Device Manager is unaware that the CAN interface is present. To solve this problem, select the interface in the Device Manager, and click on the **Remove** button. Next, click the **Refresh** button. At this point, the system rescans the installed hardware, and the CAN interface should show up without any problems. If the problem persists, contact National Instruments.
- Code 27—Windows 98/95 was unable to assign the CAN interface any resources. To solve this problem, free up system resources by disabling other unnecessary hardware so that enough resources are available for the CAN interface. The resources required for a single CAN interface are an Interrupt Request level and an 8 KB physical Memory Range (such as D0000 to D1FFF hex).

Troubleshooting Diagnostic Utility Failures

The following sections explain common error messages generated by the NI-CAN Diagnostic utility.

Memory Resource Conflict

This error occurs if the memory resource assigned to a CAN interface conflicts with the memory resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use resources that have not been reserved properly with the Device Manager. If a resource conflict exists, write down the memory

resource that caused the conflict and refer to the *Microsoft Windows User's Guide* for instructions on how to use the Device Manager to reserve memory resources for legacy boards. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

Interrupt Resource Conflict

This error occurs if the interrupt resource assigned to a CAN interface conflicts with the interrupt resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use resources that have not been reserved properly with the Device Manager. If a resource conflict exists, write down the interrupt resource that caused the conflict and refer to the *Microsoft Windows User's Guide* for instructions on how to use the Device Manager to reserve interrupt resources for legacy boards. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

NI-CAN Software Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects that it is unable to communicate correctly with the CAN hardware using the installed NI-CAN software. If you get this error, shut down your computer, restart it, and run the NI-CAN Diagnostic utility again. If the problem persists, try reinstalling the NI-CAN software.

Missing CAN Interface

If a National Instruments CAN interface is physically installed in your system, but is not listed in the NI-CAN Diagnostic utility, check the Windows Device Manager to see if Windows 98/95 has detected the hardware. For more information, refer to the *Troubleshooting Windows Device Manager Problems* section, earlier in this appendix.

CAN Hardware Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects a defect in the CAN hardware. If you get this error, write down the numeric code shown with the error, and contact National Instruments. Depending on the cause of the hardware failure, National Instruments may need to upgrade your CAN interface.

Common Questions

What do I do if my CAN hardware is listed in the Windows Device Manager with a circled X or exclamation point (!) overlaid on it?

Refer to the *Problem Shown in Device Manager* section of this appendix for specific information about what might cause this problem. If you have already completed the troubleshooting steps, fill out the forms in Appendix D, *Customer Communication*, and contact National Instruments.

How can I determine which type of CAN hardware I have installed?

Run the NI-CAN Configuration utility. To run the utility, select **Start»Settings»Control Panel»System**. Select the **Device Manager** tab in the **System Properties** dialog box. Click on the **View devices by type** radio button at the top of the sheet. If any CAN hardware is correctly installed, a **National Instruments CAN Interfaces** icon appears in the list of device types. Double-click this icon to see a list of installed CAN hardware.

How can I determine which version of the NI-CAN software I have installed?

Run the NI-CAN Diagnostic utility. To run the utility, select the **Diagnostic** item under **Start»Programs»National Instruments CAN**. The NI-CAN Diagnostic utility displays information about the version of the NI-CAN software currently installed.

What do I do if the NI-CAN Diagnostic utility fails with an error?

Refer to the *Troubleshooting Diagnostic Utility Failures* section of this appendix for specific information about what might cause the NI-CAN Diagnostic utility to fail. If you have already completed the troubleshooting steps, contact National Instruments.

How many CAN interfaces can I configure for use with my NI-CAN software for Windows 98/95?

The NI-CAN software for Windows 98/95 can be configured to communicate with up to 10 CAN interfaces.

Are interrupts required for the NI-CAN software for Windows 98/95?

Yes, one interrupt per interface is required.

How do I use an NI-CAN language interface?

For information about using NI-CAN language interfaces, refer to Chapter 2, *Developing Your Application.*

How do I use NI-CAN from within LabVIEW?

For information about using NI-CAN from within LabVIEW, refer to Chapter 2, *Developing Your Application.*

Why does the uninstall program leave some components installed?

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory, because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

C

Windows NT: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems with the NI-CAN software for Windows NT and answers some common questions.

Missing CAN Interface in the NI-CAN Configuration Utility

The NI-CAN Configuration utility contains configuration information for all of the CAN hardware it is aware of that is installed in your system. To start the NI-CAN Configuration utility, double-click on **NI-CAN Configuration under Start»Settings»Control Panel**.

If the CAN interface you are looking for is not listed under **National Instruments CAN Interfaces**, the CAN interface is not properly installed. For National Instruments CAN hardware, this problem indicates that the interface is not physically present in the system. If the interface is firmly plugged into its slot and the problem persists, contact National Instruments.

Troubleshooting Diagnostic Utility Failures

The following sections explain common error messages generated by the NI-CAN Diagnostic utility.

No Resources Assigned

This error occurs if you have not assigned resources to the CAN interface. Refer to Chapter 2, *Installation and Configuration*, in your NI-CAN getting started manual for information on assigning memory and interrupt resources to the CAN interface.

Memory Resource Conflict

This error occurs if the memory resource assigned to a CAN interface conflicts with the memory resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use the resources you assigned using the NI-CAN Configuration utility. If a resource conflict exists, use the **Resources** button in the NI-CAN Configuration utility to assign a different memory resource to the CAN interface. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

Interrupt Resource Conflict

This error occurs if the interrupt resource assigned to a CAN interface conflicts with the interrupt resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use the resources you assigned using the NI-CAN Configuration utility. If a resource conflict exists, use the **Resources** button in the NI-CAN Configuration utility to assign a different interrupt resource to the CAN interface. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

NI-CAN Software Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects that it is unable to communicate correctly with the CAN hardware using the installed NI-CAN software. If you get this error, shut down your computer, restart it, and run the NI-CAN Diagnostic utility again. If the problem persists, try reinstalling the NI-CAN software.

Missing CAN Interface

If a National Instruments CAN interface is physically installed in your system, but is not listed in the NI-CAN Diagnostic utility, check to see if the NI-CAN Configuration utility has detected the hardware. For more information, refer to the *Missing CAN Interface in the NI-CAN Configuration Utility* section, earlier in this appendix.

CAN Hardware Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects a defect in the CAN hardware. If you get this error, write down the numeric code shown with the error, and contact National Instruments. Depending on the cause of the hardware failure, National Instruments may need to upgrade your CAN interface.

Common Questions

How can I determine which type of CAN hardware I have installed?

Run the NI-CAN Configuration utility. To run the utility, select **Start»Settings»Control Panel»NI-CAN Configuration**. If any CAN hardware is correctly installed, it is listed under **National Instruments CAN Interfaces**.

How can I determine which version of the NI-CAN software I have installed?

Run the NI-CAN Diagnostic utility. To run the utility, select the **Diagnostic** item under **Start»Programs»National Instruments CAN**. The NI-CAN Diagnostic utility displays information about the version of the NI-CAN software currently installed.

What do I do if the NI-CAN Diagnostic utility fails with an error?

Refer to the *Troubleshooting Diagnostic Utility Failures* section of this appendix for specific information about what might cause the NI-CAN Diagnostic utility to fail. If you have already completed the troubleshooting steps, contact National Instruments.

How many CAN interfaces can I configure for use with my NI-CAN software for Windows NT?

The NI-CAN software can be configured to communicate with up to 10 CAN interfaces.

Are interrupts required for the NI-CAN software for Windows NT?

Yes, one interrupt per card is required.

How do I use an NI-CAN language interface?

For information about using NI-CAN language interfaces, refer to Chapter 2, *Developing Your Application.*

How do I use NI-CAN from within LabVIEW?

For information about using NI-CAN from within LabVIEW, refer to Chapter 2, *Developing Your Application.*

Why does the uninstall program leave some components installed?

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory, because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

D

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.natinst.com/support`.

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.natinst.com/worldwide`.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, Norway 32 27 73 00, Singapore 2265886,
Spain (Barcelona) 93 582 0251, Spain (Madrid) 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51,
Taiwan 02 2377 1200, United Kingdom 01635 523545

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
k-	kilo-	10^{3}

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/LS	See Low-speed CAN.
CAN data frame	Frame used to transmit the actual data of a CAN Object. The RTR bit is clear, and the data length indicates the number of data bytes in the frame.
CAN frame	In addition to fields used for error detection/correction, a CAN frame consists of an arbitration ID, an Identifier Extension, SOF and EOF bits, the RTR bit, a four-bit Data Length Code, 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

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

M

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

N

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

O

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

P

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

R

RAM	Random-access memory.
remote	Within NI-CAN, anything that exists in another node of the device network (not on the same host as the NI-CAN driver).
Remote Transmission Request (RTR) bit	This bit follows the arbitration ID in a frame, and indicates whether the frame is the actual data of the CAN Object (CAN data frame), or whether the frame is a request for the data (CAN remote frame).
request/response	Network connection in which a request is transmitted to one or more destination nodes, and those nodes send a response back to the requesting node. In industrial applications, the responding (slave) device is usually a sensor or actuator, and the requesting (master) device is usually a controller. Also known as *master/slave*.

resource	Hardware settings used by National Instruments CAN hardware, including an interrupt request level (IRQ) and an 8 KB physical memory range (such as D0000 to D1FFF hex).

S

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

U

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

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

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

W