

NI-DNET™ Programmer Reference Help

April 2006, 371106B-01

This help file is a programming reference for functions, objects, and data types in the NI-DNET software for Windows. This help file assumes that you are already familiar with Windows.

For more information about this help file, refer to the following topics:

Using Help

Related Documentation

Glossary

Important Information

Technical Support and Professional Services

To comment on National Instruments documentation, refer to the <u>National Instruments Web site</u>.

© 2004–2006 National Instruments Corporation. All rights reserved.

How to Use the Documentation Set

Use the installation guide to install and configure your DeviceNet hardware and NI-DNET software.

Use the *NI-DNET User Manual* to learn the basics of NI-DNET and how to develop an application. The user manual also contains information about DeviceNet hardware.

Use this *NI-DNET Programmer Reference Help* for specific information about each NI-DNET function and object.

Related Documentation

Most NI-DNET manuals also are available as PDFs. You must have Adobe Reader with Search and Accessibility 5.0.5 or later installed to view the PDFs. Refer to the <u>Adobe Systems Incorporated Web site</u> at www.adobe.com to download Adobe Reader. Refer to the <u>National Instruments Product Manuals Library</u> at ni.com/manuals for updated documentation resources.

The following documents contain information that you may find helpful as you use this help file:

- *CAN Specification Version 2.0*, 1991, Robert Bosch Gmbh., Postfach 500, D-7000 Stuttgart 1
- DeviceNet Specification, Volumes 1 and 2, Version 2.0, Open DeviceNet Vendor Association
- LabVIEW online reference
- Microsoft Win32 Software Development Kit (SDK) online help

Using Help

Conventions

Navigating Help

Searching Help

Printing Help File Topics

Conventions

This help file uses the following formatting and typographical conventions:

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 in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

green Underlined text in this color denotes a link to a help topic, help file, or Web address.

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

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

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

Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- **Hide**—Hides the navigation pane from view.
- Locate—Locates the currently displayed topic in the Contents tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- **Forward**—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

Wildcards

You also can search using asterisk (*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



Note Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

Nested Expressions

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

Boolean Expressions

Click the **•** button to add Boolean expressions to a search. The following Boolean operators are available:

- AND (default)—Returns topics that contain both search terms. You
 do not need to specify this operator unless you are using nested
 expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

Search Options

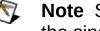
Use the following checkboxes on the **Search** tab to customize a search:

- **Search previous results**—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- Match similar words—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- **Search titles only**—Searches only in the titles of topics.

Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

- 1. Right-click the book.
- 2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
- 3. Select the **Print the selected heading and all subtopics** option.



Note Select Print the selected topic if you want to print the single topic you have selected in the **Contents** tab.

4. Click the **OK** button.

Printing PDF Documents

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

NI-DNET Data Types

This topic describes the data types used by NI-DNET functions and objects.

The NI-DNET data types provide consistency for various programming environments and facilitate access to the DeviceNet network. In general, all NI-DNET data types begin with NCTYPE_.

The following table lists each NI-DNET data type, its equivalent data type in ANSI C, LabVIEW, and DeviceNet, and a brief description.

NI-DNET Data Types

NI-DNET Data Type	ANSI C	LabVIEW	DeviceNet	[
NCTYPE_type_P	NCTYPE_type *	N/A	N/A	Pointe with ty
NCTYPE_INT8	signed char	I8	SINT	8-bit s
NCTYPE_INT16	signed short	I16	INT	16-bit
NCTYPE_INT32	signed long	I32	DINT	32-bit
NCTYPE_UINT8	unsigned char	U8	USINT	8-bit u
NCTYPE_UINT16	unsigned short	U16	UINT	16-bit
NCTYPE_UINT32	unsigned long	U32	UDINT	32-bit
NCTYPE_BOOL	unsigned char	TF (Boolean)	BOOL	Boolea ANSI (NC_TI NC_F/ used f
NCTYPE_STRING	char *, array of characters terminated by null character \0	abc (string)	STRING	ASCII
NCTYPE_REAL	float	SGL	REAL	32-bit
NCTYPE_LREAL	double	DBL	LREAL	64-bit
NCTYPE_ANY_P	void *	N/A	N/A	Refere of unk in case data ty

				depen particı
NCTYPE_OBJH	unsigned long	Type definition ncObjHandle.ctl (U32)	N/A	Handle NI-DN to ncO ncOper ncOper
NCTYPE_VERSION	unsigned long	U32	N/A	Versio minor, beta v are en unsigr from h byte. L encod equiva is 26, Versio hexad and Bobeta 7 010402
NCTYPE_DURATION	unsigned long	U32	N/A	Time c indicat between Time is ms inc examp 10,000 constan NC_D (0) is constan NC_D (1) is constan NC_D (2) is constan NC_D (3) is constan NC_D (4) is constan NC_D (5) is duratic nex) is duratic

NCTYPE_ATTRID	unsigned long	U32	N/A	Identif access attribu DNET (not at Device Refer Object
NCTYPE_OPCODE	unsigned long	U32	N/A	Opera with no function
NCTYPE_STATE	unsigned long	U32	N/A	Object as 32- for eac inform ncWait
NCTYPE_STATUS	signed long	I32	N/A	For An repres returne DNET to ncSt more i LabVII functic standa for sta

NI-DNET Functions

The following topics list all NI-DNET functions and describe the purpose, format, parameters, and return status for each function.

All NI-DNET functions are reentrant to achieve good multitasking performance. Unless otherwise stated, each NI-DNET function suspends execution of your program until it completes.

<u>Using the Function Descriptions</u>

List of NI-DNET Functions

Using the Function Descriptions

The description of each function is structured as follows:

Purpose

States the function's purpose.

Format

Describes the function's format for the LabVIEW and C (including C++) programming languages.

Input

Lists the function's input parameters (values passed into the function).

Output

Lists the function's output parameters (values passed out of the function).

Function Description

Provides details about the function's purpose and effect.

Parameter Description

Provides details about each input/output parameter, including allowed values and their meanings.

Examples

Each function description includes sample LabVIEW and C code showing how to use the function. For more detailed examples, refer to the *NI-DNET User Manual* for information regarding the location of example programs for LabVIEW and C.

List of NI-DNET Functions

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

NI-DNET Functions

Function	Purpose
EasyIOClose(Easy IO Close)	Close multiple NI-DNET objects (LabVIEW only)
EasyIOConfig (Easy IO Config)	Configure and open multiple NI-DNET objects (LabVIEW only)
ncCloseObject (Close)	Close an NI-DNET object
ncConvertForDnetWrite (Convert for DeviceNet Write)	Convert an appropriate LabVIEW data type for writing data bytes on the DeviceNet network
ncConvertFromDnetRead (Convert From DeviceNet Read)	Convert data from the DeviceNet network into an appropriate LabVIEW data type
ncCreateNotification (Create Notification)	Create a notification callback for an object (C only)
ncGetDnetAttribute (Get DeviceNet Attribute)	Get an attribute value from a DeviceNet device using an Explicit Messaging Object
ncGetDriverAttr (Get Driver Attribute)	Get the value of an attribute in the NI-DNET driver
ncOpenDnetExplMsg (Open DeviceNet Explicit Messaging)	Configure and open an NI-DNET Explicit Messaging Object
ncOpenDnetIntf (Open DeviceNet Interface)	Configure and open an NI-DNET Interface Object
ncOpenDnetIO (Open DeviceNet I/O)	Configure and open an NI-DNET I/O Object
ncOperateDnetIntf (Operate DeviceNet Interface)	Perform an operation on an NI-DNET Interface Object
ncReadDnetExplMsg (Read DeviceNet Explicit Message)	Read an explicit message response from an Explicit Messaging Object

ncReadDnetIO (Read DeviceNet I/O)	Read input from an I/O Object
ncSetDnetAttribute (Set DeviceNet Attribute)	Set an attribute value for a DeviceNet device using an Explicit Messaging Object
ncSetDriverAttr (Set Driver Attribute)	Set the value of an attribute in the NI-DNET driver
ncStatusToString (Status to String)	Convert status returned from an NI-DNET function into a descriptive string (C only)
ncWaitForState (Wait for State)	Wait for one or more states to occur in an object
ncWriteDnetExplMsg (Write DeviceNet Explicit Message)	Write an explicit message request using an Explicit Messaging Object
ncWriteDnetIO (Write DeviceNet I/O)	Write output to an I/O Object

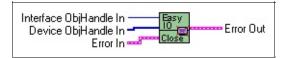
EasylOClose (Easy IO Close)

Purpose

Close multiple NI-DNET objects in one call.

Format

LabVIEW



C

Not applicable

Input

In

Interface ObjHandle Object handle of an open Interface Object, returned

from either Easy IO Config or Open DeviceNet

Interface function

Device ObjHandle In Array of I/O and/or Explicit Messaging object

handles

Error in NI-DNET Error Cluster input

Output

Error out

NI-DNET Error Cluster output

Function Description

EasyIOClose stops the Interface Object, closes all the object handles passed in the Device ObjHandle In parameter, and then closes the Interface Object. You normally call EasyIOClose near the end of your application to ensure that all objects are properly deallocated.

EasyIOClose accepts Interface ObjHandle In and Device ObjHandle In as input parameters. You pass the outputs from EasyIOConfig as inputs to EasyIOClose.

Internally, the EasyIOClose function makes use of OperateDeviceNetInterface.vi (ncOperateDnetIntf) and CloseObject.vi (ncCloseObject). To learn more about these functions, refer to the corresponding function description sections.

Parameter Descriptions

Interface ObjHandle In

•	Contains an interface object handle returned from the Easy IO Config or Open DeviceNet Interface function.
Values	The encoding of object handle is internal to NI-DNET.

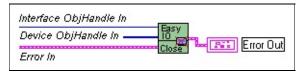
Device ObjHandle In

•	Array of I/O object handles to be closed. You pass in the
	array returned from Easy IO Config.
Values	The encoding of object handles is internal to NI-DNET.

Examples

LabVIEW

Close Interface Object and I/O Objects opened with Easy IO Config.



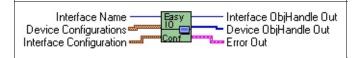
EasylOConfig (Easy IO Config)

Purpose

Configure and open an NI-DNET Interface Object and multiple NI-DNET I/O Objects.

Format

LabVIEW



C

Not applicable

Input

Device Array of I/O Object configuration clusters

Configurations MAC ID of the remote device

DeviceMacId Type of I/O connection
ConnectionType Number of input bytes
InputLength Number of output bytes

OutputLength Expected rate of I/O message (packet) production

ExpPacketRate

IntfMacId Baud rate

BaudRate Communication scheme for all polled I/O

PollMode connections

Output

Interface Object handle you use with all subsequent function calls

ObjHandle Out for the Interface Object

Device Array of object handles you index to reference a

ObjHandle Out particular I/O Object

Error out NI-DNET Error Cluster output

Function Description

EasyIOConfig configures, opens, and starts an Interface and multiple I/O Objects, and returns object handles for the newly created objects.

Internally, the EasyIOConfig function makes use of ncOpenDnetIntf, ncOpenDnetIO, ncOperateDnetIntf, and ncWaitForState. If you are not familiar with the input clusters mentioned above, refer to ncOpenDnetIntf and/or ncOpenDnetIO parameter descriptions before reading this section. For more details on any of these functions, refer to the corresponding function descriptions.

Use EasyIOConfig to open multiple devices (I/O connections) with one VI call. This high-level function accepts Interface Configuration and an array of Device Configurations as its inputs. The Device Configurations can contain any number of I/O connections that you want to open. Remember, however, that you can only have one instance of a particular connection per device. For example, you cannot open two poll connections on the same device. Similarly, opening COS and cyclic connections simultaneously on a device will result in an error, since these two connections are mutually exclusive.

The relationship between expected packet rate (EPR) and the PollMode parameter of the Interface Object is the same as discussed in the ncOpenDnetIntf and ncOpenDnetIO function descriptions. For example, if you configure the Interface Object in Scanned mode, you must configure all the strobe connections with the same EPR and all the poll connections with either the same EPR value or an integer multiple of it. If this is not the case, you will see an Inconsistent Parameter error.

Since the EasyIOConfig function also starts the interface, a call to ncOperateDnetIntf (for Start) is only needed if the communication needs to be interrupted in the middle of your application to set some driver attributes for an object. To do so, call ncOperateDnetIntf with Stop as the Opcode after calling EasyIOConfig, make necessary calls to ncSetDriverAttr, and then call ncOperateDnetIntf with Start as the Opcode to restart the communication.

To open an Explicit Messaging Object, call ncOpenDnetExplMsg separately after a call to EasyIOConfig.



Note For any NI-DNET LabVIEW application, make sure that all

the open calls are matched by an equal number of close calls. For example, if you have called the Open DeviceNet Interface function twice, you must call the Close Object function twice as well, passing in the handles returned from the open interface calls. Also, to ensure proper closure of all NI-DNET objects, create your own stop button to stop your application, instead of using the LabVIEW stop button from the menu bar.

Parameter Descriptions

Interface Name

Name of the DeviceNet interface as an ASCII string with format "DNETx", where x is a decimal number starting at zero that indicates which DeviceNet interface is being used. You associate DeviceNet interface names with physical ports using Measurement and Automation Explorer (MAX).
"DNET0", "DNET1", "DNET31" In LabVIEW, you select the interface name from an enumerated list.

Device Configurations

Array of NI-DNET I/O Object configuration clusters. For a description of individual elements within the I/O cluster, refer to the ncOpenDnetIO parameter description.
Refer to the ncOpenDnetIO input parameters description for value range applicable to each configuration parameter.

Interface Configuration

Configuration cluster for NI-DNET Interface Object. For a description of individual elements within the interface cluster, refer to the ncOpenDnetIntf parameter description.
Refer to the ncOpenDnetIntf input parameters description for allowed values for each cluster element.

Interface ObjHandle Out

	If the Easy IO Config function is successful, a handle to the newly opened Interface Object is returned in Interface ObjHandle Out. This handle is used with all subsequent function calls for that Interface Object.
Values	The encoding of object handles is internal to NI-DNET.

Device ObjHandle Out

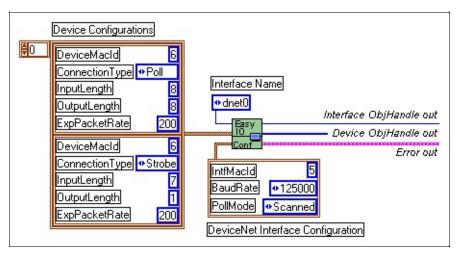
Description	If the Easy IO Config function is successful, an array of I/O
	Object handles is returned in Device ObjHandle Out. This
	array can be indexed to retrieve individual I/O handles for
	data read and write.

Values	The encoding of object handles is internal to NI-DNET.

Examples

LabVIEW

Open Interface Object "DNET0" using baud rate 125000, MAC ID 5, and poll mode Scanned. Open two I/O Objects, with MAC ID 6 and 9, and start the communication.



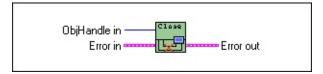
ncCloseObject (Close)

Purpose

Close an NI-DNET object.

Format

LabVIEW



C

NCTYPE_STATUS ncCloseObject(NCTYPE_OBJH ObjHandle)

Input

Object handle of an open Interface Object, Explicit Messaging Object, or I/O Object ObjHandle

Output

None

Function Description

The ncCloseObject function closes an NI-DNET object when it no longer needs to be in use, such as when the application is about to terminate. When an object is closed, NI-DNET stops all pending operations for the object, and you can no longer use the ObjHandle in your application.

If the object specified by ObjHandle has a notification pending, ncCloseObject disables the notification by implicitly calling ncCreateNotification with DesiredState zero.

When ncCloseObject has been called for all open NI-DNET objects, NI-DNET stops all DeviceNet communication (ncCloseObject issues an implicit call to ncOperateDnetIntf with Opcode NC_OP_STOP).

Parameter Descriptions

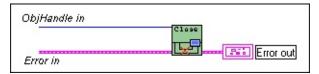
ObjHandle

	ObjHandle must contain an object handle returned from the ncOpenDnetIntf, ncOpenDnetExplMsg, or ncOpenDnetIO function.
Values	The encoding of ObjHandle is internal to NI-DNET.

Examples

LabVIEW

Close an NI-DNET object.



C

Close an NI-DNET object.

NCTYPE_STATUS status;
NCTYPE_OBJH objh;
status = ncCloseObject (objh);

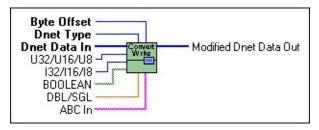
ncConvertForDnetWrite (Convert For DeviceNet Write)

Purpose

Convert an appropriate LabVIEW data type for writing data bytes on the DeviceNet network.

Format

LabVIEW



C

Not applicable, but see **Examples** at the end of this section

Input

DnetData in Initial data bytes to write on the DeviceNet network

DnetType DeviceNet data type to convert into

ByteOffset Byte offset of the DeviceNet member to convert into

8[TF] in LabVIEW array of 8 TF to convert from

I32/I16/I8 in LabVIEW I32, I16, or I8 to convert from

U32/U16/U8 in LabVIEW U32, U16, or U8 to convert from

DBL/SGL in LabVIEW DBL or SGL to convert from

abc in LabVIEW string to convert from

Output

DnetData out DeviceNet data bytes (with member inserted)

Function Description

Many fundamental differences exist between the encoding of a DeviceNet data type and its equivalent data type in LabVIEW. For example, for a 32-bit integer, the DeviceNet DINT data type uses Intel byte ordering (lowest byte first), and the equivalent LabVIEW I32 data type uses Motorola byte ordering (highest byte first).

ncConvertForDnetWrite takes an initial sequence of bytes to write on the DeviceNet network, and given the byte offset and DeviceNet data type for a specific data member, converts an appropriate LabVIEW data type for placement into those data bytes. You provide initial data bytes using DnetData in, convert a LabVIEW data type for each data member changed by your LabVIEW program (possibly replacing all initial bytes with LabVIEW data), then write the bytes onto the DeviceNet network.

You typically use ncConvertForDnetWrite with the following NI-DNET functions:

- ncWriteDnetIO—Convert a LabVIEW data type for placement into the output assembly.
- ncSetDnetAttribute—Convert a LabVIEW data type to set as the attribute value.
- ncWriteDnetExplMsg—Convert a LabVIEW data type for placement into the service request.

Since DeviceNet data types are similar to C language data types, C programming does not need a function like ncConvertForDnetWrite. By using standard C language pointer manipulations, you can convert an appropriate C language data type for writing as a DeviceNet data member. For more information about converting C language data types, refer to the Examples at the end of this section.

Parameter Descriptions

DnetData in

Description	Initial data bytes to write on the DeviceNet network. These data bytes are normally created as a constant array of U8, then given valid default values. If you need to convert multiple DeviceNet data members, you can wire this input terminal from the DnetData out output terminal of a previous use of this function.
	If you replace all initial data bytes using this function, the default values are unimportant, and you can leave them as zero.
Values	Initial data bytes to write on the DeviceNet network or
	DnetData out output terminal of a previous use of this function

DnetType

Description	An enumerated list from which you choose the DeviceNet data type to convert into. For each DeviceNet data type, the appropriate LabVIEW data type is listed in parentheses.
	When you select the DeviceNet data type BOOL, ncConvertForDnetWrite converts the byte indicated by ByteOffset from an array of eight LabVIEW Booleans. You can index into this array to change specific Boolean members. The Boolean at index zero is the least significant bit (bit 0), the Boolean at index one is the next least significant (bit 1), and so on.
Values	BOOL (8[TF]) SINT (I8) INT (I16) DINT (I32) USINT (U8) UINT (U16)

UDINT (U32)
REAL (SGL)
LREAL (DBL)
SHORT_STRING (abc)
STRING (abc)

ByteOffset

Description	Byte offset of the DeviceNet member to convert into. For the DeviceNet data member you want to replace, this is the byte offset in DnetData in where the member begins. Byte offsets start at zero.
	You can find information on the format of your DeviceNet data in the following functions:
	 ncWriteDnetIO—Specification for your device's output assembly.
	 ncSetDnetAttribute—Data type of the attribute. Unless the attribute's DeviceNet data type is a structure or array, the value for ByteOffset is always 0.
	 ncWriteDnetExplMsg—Specification for the service data of the explicit message request.
Values	0 to 255

8[TF] in

Description	If the selected DnetType is BOOL, this input terminal provides the LabVIEW data to convert into a DeviceNet data member. The LabVIEW data type for this input terminal is an array of eight LabVIEW Booleans, indicated as 8[TF]. You can index into this array to change specific Boolean members. The Boolean at index zero is the least significant
	members. The Boolean at index zero is the least significant bit (bit 0), the Boolean at index one is the next least significant (bit 1), and so on.
Values	LabVIEW data to convert into a DeviceNet data member

I32/I16/I8 in

Description	If the selected DnetType is SINT, INT, or DINT, this input
	terminal provides the LabVIEW data to convert into a
	DeviceNet data member. Although the LabVIEW data type

	for this input terminal is I32, it can be coerced automatically from I16 or I8.
Values	LabVIEW data to convert into a DeviceNet data member

U32/U16/U8 in

	If the selected DnetType is USINT, UINT, or UDINT, this input terminal provides the LabVIEW data to convert into a DeviceNet data member. Although the LabVIEW data type for this input terminal is U32, it can be coerced automatically from U16 or U8.
Values	LabVIEW data to convert into a DeviceNet data member

DBL/SGL in

	If the selected DnetType is REAL or LREAL, this input terminal provides the LabVIEW data to convert into a DeviceNet data member. Although the LabVIEW data type for this input terminal is DBL, it can be coerced automatically from SGL.
Values	LabVIEW data to convert into a DeviceNet data member

abc in

	If the selected DnetType is SHORT_STRING or STRING, this input terminal provides the LabVIEW data to convert into a DeviceNet data member. The LabVIEW data type for this input terminal is abc.
Values	LabVIEW data to convert into a DeviceNet data member

DnetData out

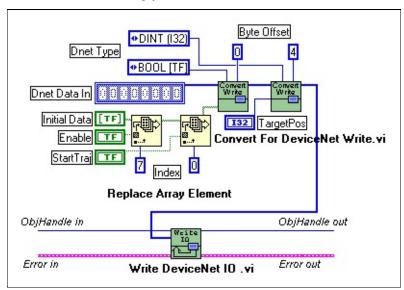
·	DeviceNet data bytes (with member inserted). These data bytes are written on the DeviceNet network using the ncWriteDnetIO, ncSetDnetAttribute, or ncWriteDnetExplMsg function. If you need to convert multiple DeviceNet data members, you can also wire this output terminal into the DnetData in input terminal of a subsequent use of this function.
Values	Data input terminal of ncWriteDnetIO or AttrData input terminal of ncSetDnetAttribute

or
ServData input terminal of ncWriteDnetExplMsg
or
DnetData in input terminal of a subsequent use of this
function

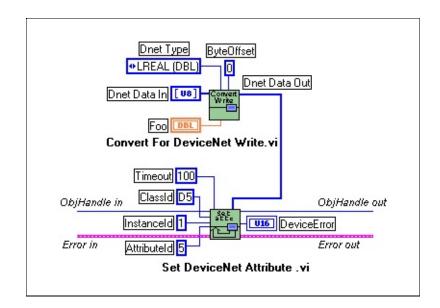
Examples

LabVIEW

1. Use ncWriteDnetIO to write Command Assembly 1 to a Position Controller. In this output assembly, the byte at offset 0 consists of 8 BOOL and the bytes at offset 4–7 consist of a Target Position of type DINT. Use ncConvertForDnetWrite to convert appropriate LabVIEW data types for these DeviceNet data members.



2. Set an attribute Foo using the ncSetDnetAttribute function. The attribute Foo is contained in an object with class ID D5 hex, instance ID 1, attribute ID 5, and its DeviceNet data type is LREAL. Use ncConvertForDnetWrite to convert the appropriate LabVIEW data type for Foo.



C

1. Demonstrate the same conversions as LabVIEW example 1.

```
NCTYPE UINT8 data[8];
NCTYPE UINT8 I;
NCTYPE INT32
                   TargetPos;
                                 /* DINT */
NCTYPE BOOL
                   Enable;
                                 /* BOOL */
                   StartTraj; /* BOOL */
NCTYPE BOOL
 /* Initialize default values of zero. */
for (I = 0; I < 8; I++)
  data[I] = 0;
 /* If Enable is true, set bit 7 of byte 0. If StartTraj is
 true, set bit 0 of byte 0. */
if (Enable == NC TRUE)
  data[0] = 0x80;
if (StartTraj == NC_TRUE)
  data[0] = 0x01;
 /* Take the address of the data byte at offset 4, cast that
  address to point to the appropriate C language data type, then
  dereference the pointer in order to store the value. */
*(NCTYPE_INT32 *)(&(data[4])) = TargetPos;
status = ncWriteDnetIO(objh, sizeof(data), data);
```

2. Demonstrate the same conversion as LabVIEW example 2.

NCTYPE_LREAL foo;

/* Conversion is performed automatically simply by passing in a pointer to the appropriate C language data type. */ foo = 354654.4543;

status = ncSetDnetAttribute(objh, 0xD5, 0x01, 0x05, 100, sizeof(foo), &foo);

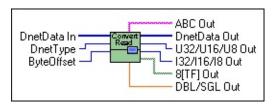
ncConvertFromDnetRead (Convert From DeviceNet Read)

Purpose

Convert data read from the DeviceNet network into an appropriate LabVIEW data type.

Format

LabVIEW



C

Not applicable, but see **Examples** at the end of this section

Input

DnetData in Data bytes read from the DeviceNet network

DnetType DeviceNet data type to convert from

ByteOffset Byte offset of the DeviceNet member to convert

Output

DnetData out DeviceNet data bytes (unchanged)

8[TF] out Converted LabVIEW array of 8 TF

I32/I16/I8 out Converted LabVIEW I32, I16, or I8

U32/U16/U8 out Converted LabVIEW U32, U16, or U8

DBL/SGL out Converted LabVIEW DBL or SGL

abc out Converted LabVIEW string

Function Description

Many fundamental differences exist between the encoding of a DeviceNet data type and its equivalent data type in LabVIEW. For example, for a 32-bit integer, the DeviceNet DINT data type uses Intel byte ordering (lowest byte first), and the equivalent LabVIEW I32 data type uses Motorola byte ordering (highest byte first).

ncConvertFromDnetRead takes a sequence of bytes read from the DeviceNet network, and given the byte offset and DeviceNet data type for a specific data member in those bytes, converts that DeviceNet data member into an appropriate LabVIEW data type.

You typically use ncConvertFromDnetRead with the following NI-DNET functions:

- ncReadDnetIO—Convert a member of the input assembly to its LabVIEW data type.
- ncGetDnetAttribute—Convert the attribute to its LabVIEW data type.
- ncReadDnetExplMsg—Convert a member in the service response to its LabVIEW data type.

Since DeviceNet data types are similar to C language data types, C programming does not need a function like ncConvertFromDnetRead. By using standard C language pointer manipulations, you can convert a DeviceNet data member into its appropriate C language data type. For more information about converting DeviceNet data members into C language data types, refer to the Examples at the end of this section.

Parameter Descriptions

DnetData in

Description	Data bytes read from the DeviceNet network. These data bytes are read from the DeviceNet network using ncReadDnetIO, ncGetDnetAttribute, or ncReadDnetExplMsg. If you need to convert multiple DeviceNet data members, you can wire this input terminal from the DnetData out output terminal of a previous use of this function.
Values	Data output terminal of ncReadDnetIO or AttrData output terminal of ncGetDnetAttribute or ServData output terminal of ncReadDnetExplMsg or DnetData out output terminal of a previous use of this function

DnetType

Description	An enumerated list from which you select the DeviceNet data type to convert. For each DeviceNet data type, the list displays the resulting LabVIEW data type in parentheses.
	When you select the DeviceNet data type BOOL, ncConvertFromDnetRead converts the byte indicated by ByteOffset into an array of eight LabVIEW Booleans. You can index into this array to use specific Boolean members. The Boolean at index zero is the least significant bit (bit 0), the Boolean at index one is the next least significant (bit 1), and so on.
Values	BOOL (8[TF]) SINT (I8) INT (I16) DINT (I32) USINT (U8) UINT (U16) UDINT (U32) REAL (SGL)

LREAL (DBL)
SHORT_STRING (abc)
STRING (abc)

ByteOffset

Description	Byte offset of the DeviceNet member to convert. For the DeviceNet data member you want to convert, this is the byte offset in DnetData in where the member begins. Byte offsets start at zero. You can find information on the format of your DeviceNet data in the following functions:
	 ncReadDnetIO—Specification for your device's input assembly. ncGetDnetAttribute—Data type of the attribute. Unless the attribute's DeviceNet data type is a structure or array, the value for ByteOffset is always 0. ncReadDnetExplMsg—Specification for the service data of the explicit message response.
Values	0 to 255

DnetData out

	DeviceNet data bytes (unchanged). The data bytes of DnetData in are passed through the VI to this output terminal unchanged. To convert another DeviceNet data member, this data can be passed on to another call to this function.
	the data can be passed on to another can to the fariotion.
Values	Same as DnetData in

8[TF] out

Description	If the selected DnetType is BOOL, this output terminal provides the converted DeviceNet data member. The LabVIEW data type for this output terminal is an array of eight LabVIEW Booleans, indicated as 8[TF]. You can index into this array to use specific Boolean members. The Boolean at index zero is the least significant bit (bit 0), the Boolean at index one is the next least significant (bit 1), and
	so on.

Values	Converted DeviceNet data member
--------	---------------------------------

I32/I16/I8 out

	If the selected DnetType is SINT, INT, or DINT, this output terminal provides the converted DeviceNet data member. Although the LabVIEW data type for this output terminal is I32, it can be coerced automatically to I16 or I8.
Values	Converted DeviceNet data member

U32/U16/U8 out

	If the selected DnetType is USINT, UINT, or UDINT, this output terminal provides the converted DeviceNet data member. Although the LabVIEW data type for this output terminal is U32, it can be coerced automatically to U16 or U8.
Values	Converted DeviceNet data member

DBL/SGL out

	If the selected DnetType is REAL or LREAL, this output terminal provides the converted DeviceNet data member. Although the LabVIEW data type for this output terminal is DBL, it can be coerced automatically to SGL.
Values	Converted DeviceNet data member

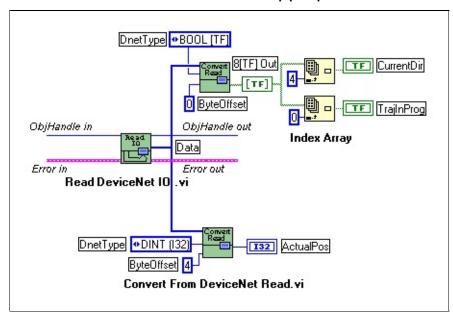
abc out

·	If the selected DnetType is SHORT_STRING or STRING, this output terminal provides the converted DeviceNet data member. The LabVIEW data type for this output terminal is abc.
Values	Converted DeviceNet data member

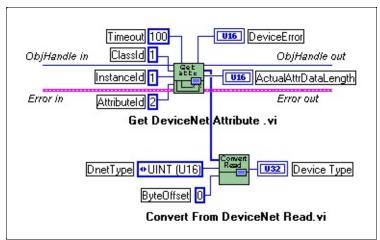
Examples

LabVIEW

 Use ncReadDnetIO to read Response Assembly 1 from a Position Controller. In this input assembly, the byte at offset 0 consists of 8 BOOL, and the bytes at offset 4–7 consist of an Actual Position of type DINT. Use ncConvertFromDnetRead to convert these DeviceNet data members into appropriate LabVIEW data types.



2. Get the Device Type attribute using the ncGetDnetAttribute function. The Device Type is contained in the Identity Object (class ID 1, instance ID 1, attribute ID 2), and its DeviceNet data type is UINT. Use ncConvertFromDnetRead to convert the Device Type into an appropriate LabVIEW data type.



1. Demonstrate the same conversions as LabVIEW example 1.

```
NCTYPE UINT8 data[8];
NCTYPE INT32 ActualPos:
                                /* DINT */
                 CurrentDir:
                                /* BOOL */
NCTYPE BOOL
NCTYPE BOOL TrailnProg;
                                /* BOOL */
status = ncReadDnetIO(objh, sizeof(data), data);
 /* Take the address of the data byte at offset 4, cast that
 address to point to the appropriate C language data type, then
 dereference the pointer. */
ActualPos = *(NCTYPE_INT32 *)(&(data[4]));
 /* If bit 4 of byte 0 is set, then CurrentDir is true. If bit
 0 of byte 0 is set, the TrajInProg is true. */
CurrentDir = (data[0] \& 0x10)? NC TRUE : NC FALSE;
TraiInProg = (data[0] \& 0x01)? NC TRUE : NC FALSE;
```

2. Demonstrate the same conversion as LabVIEW example 2.

ncCreateNotification (Create Notification) Purpose

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

Format

LabVIEW

Not applicable

С

NCTYPE_STATUS

Input

ObjHandle Object handle of an open Explicit Messaging Object or

I/O Object

DesiredState States for which notification is called

Timeout Number of milliseconds to wait for one of the desired

states

RefData Pointer to user-specified reference data

Callback Address of your callback function

Output

None

Function Description

ncCreateNotification creates a notification callback for the object specified by ObjHandle. The NI-DNET driver uses the notification callback to communicate state changes to your application. The ncCreateNotification function does not apply to LabVIEW programming.

You commonly use ncCreateNotification to receive notifications when new input data is available for an I/O Object. Within your notification callback function, you call ncReadDnetIO to read the new input data, perform any needed calculations for that data, call ncWriteDnetIO to provide output data, then return from the callback function.

You normally use ncCreateNotification when you want to let other code to execute while waiting for NI-DNET states, especially when the other code does not call NI-DNET functions. If you do not need such background execution, ncWaitForState offers better overall performance. You cannot use ncWaitForState at the same time as ncCreateNotification.

This function is not supported for Visual Basic 6.

The Status parameter of your callback function indicates any error detected by NI-DNET. You should always check this Status parameter prior to checking the CurrentState parameter of your callback function.

When ncCreateNotification returns successfully, NI-DNET calls your notification callback function whenever one of the states specified by DesiredState occurs in the object. If DesiredState is 0, NI-DNET disables notifications for the object specified by ObjHandle.

Parameter Descriptions

ObjHandle

•	ObjHandle must contain an object handle returned from
	ncOpenDnetExplMsg or ncOpenDnetIO.
Values	The encoding of ObjHandle is internal to NI-DNET.

DesiredState

Description States for which notification is called. So that notification can be enabled for multiple states simultaneously, a single bit represents each state. For example, if NI-DNET provides states with values of hex 1 and hex 4. DesiredState of hex 5 enables notification for both states.

ReadAvail for the I/O Object

For the I/O Object, the ReadAvail state sets when a new input message is received from the network. The ReadAvail state clears when you call ncReadDnetIO. For example, for a change-of-state (COS) I/O connection, the notification occurs when a COS input message is received.

The typical behavior for your callback function is to call ncReadDnetIO to read the new input data, perform any calculations needed, call ncWriteDnetIO to provide output data, then return from the callback function.

ReadAvail for the Explicit Messaging Object

For the Explicit Messaging Object, the ReadAvail state sets when an explicit message response is received from the network. The ReadAvail state clears when you call ncReadDnetExplMsg. An explicit message response is received only after you send an explicit message request using ncWriteDnetExplMsg.

Although using a notification for an explicit message response allows for execution of other code while waiting, it is often more straightforward to use the following sequence of calls: ncWriteDnetExplMsg, ncWaitForState, and ncReadDnetExplMsg. This is the sequence used internally by ncGetDnetAttribute and ncSetDnetAttribute.

The ReadAvail state is not needed when using the explicit messaging functions ncGetDnetAttribute and ncSetDnetAttribute because both of these functions wait for the explicit message response internally.

Established for the Explicit Messaging Object

For the Explicit Messaging Object, the Established state is clear (not established) before you start communication using ncOperateDnetIntf. After you start communication, the Established state remains clear until the explicit message connection has been successfully established with the remote DeviceNet device. After the explicit message connection has been established, the Established state sets and remains set for as long as the explicit message connection is open.

Until the Established state is set for the Explicit Messaging Object, all calls to ncGetDnetAttribute, ncSetDnetAttribute, or ncWriteDnetExplMsg return the error CanErrNotStarted. Before you call any of these functions in your application, you must first wait for the Established state to set.

After the Established state is set, unless communication problems occur with the device (CanErrFunctionTimeout), it remains set until you stop communication using ncOperateDnetIntf.

While waiting for one of the above states, if an error occurs (such as a communication error or an initialization error), the notification returns immediately with the appropriate error code. For example, if you call ncCreateNotification with DesiredState of ReadAvail, the notification function will return when data is available for a read, or when a DeviceNet communication error (such as connection timeout) is detected.

Values

A combination of the following bit values:

1 hex (ReadAvail state, constant NC_ST_READ_AVAIL)

8 hex (Established, constant NC_ST_ESTABLISHED)

In the LabWindows™/CVI™ function panel, to facilitate combining multiple states, you can select a combination from an enumerated list of all valid combinations. This list contains the names of each state in the combination, such as ReadAvail or Established.

Timeout

Description	Number of milliseconds to wait for one of the desired states. If the timeout expires before one of the desired states occurs, your notification function is called with CurrentState of 0 and Status of CanErrFunctionTimeout. Use the special timeout value of FFFFFFFF hex to wait indefinitely.
Values	1 to 200000 or FFFFFFFF hex (infinite duration, constant NC_DURATION_INFINITE)

RefData

Description	RefData provides a pointer that is passed to all calls of your notification callback function. It is typically used to provide the address of globally declared reference data for use within the notification callback. For example, for the ReadAvail state, RefData is often the data buffer which you pass to ncReadDnetIO to read available data. If the notification callback does not need reference data, you can set RefData to NULL.
Values	Pointer to any globally declared data variable or NULL

Callback

•	This is the address of a callback function within your
	application source code. Within the code for the callback
	function, you can call any of the NI-DNET functions except
	for ncCreateNotification and ncWaitForState.

Declare this function using the following C language prototype.

In the declaration for your callback, the constant _NCFUNC_ is required for your compiler to declare the function such that it can be called by the NI-DNET device driver.

Parameter descriptions for Callback

ObjHandle

Object handle originally passed to ncCreateNotification. This identifies the object generating the notification, which is useful when you use the same callback function for multiple objects.

CurrentState

Current state of the object. If one of the desired states occurs, it provides the current value of the ReadAvail and Established states. If the Timeout expires before one of the desired states occurs, it has the value 0.

Status

Current status of the object. If one of the desired states occurs, it has the value 0 (DnetSuccess). If the Timeout expires before one of the desired states occurs, it has the value BFF62001 hex (CanErrFunctionTimeout).

RefData

Pointer to your reference data as originally passed to ncCreateNotification.

Return Value from Callback

The value you return from the callback indicates the desired states to re-enable for notification. If you want to continue to

receive notifications, return the same value as the original DesiredState parameter. If you no longer want to receive notifications, return a value of 0.

If you return a nonzero value from the callback, and one of those states is still set, the callback is invoked again immediately after you return. For example, if you return ReadAvail from the callback without calling ncReadDnetIO to read the available data, the callback is invoked again. Information Specific to LabWindows/CVI

When the NI-DNET device driver calls your notification callback, it does so in a separate thread within the LabWindows/CVI process. Your application's front panel indicators and controls can only be accessed within the main thread of the LabWindows/CVI process. Although you can call NI-DNET functions and perform generic C calculations in your notification callback, you cannot call LabWindows/CVI functions which access the front panel (the User Interface Library). To use the LabWindows/CVI User Interface Library, save any data needed for front panel indicators using global variables, then register a deferred callback using the LabWindows/CVI PostDeferredCall function. Since a LabWindows/CVI deferred callback executes in the main thread of the LabWindows/CVI process, you can call any LabWindows/CVI function, including the User Interface Library.

Information Specific to Microsoft, Borland, and Other C Compilers

When the NI-DNET device driver calls your notification callback, it does so in a separate thread within your process. Therefore, it has access to any process global data, but not thread local data. If your callback function needs to access global variables, you must protect that access using synchronization primitives (such as semaphores) because your callback is running in a different thread context. For an explanation of these concepts and other multithreading issues, refer to the online help of the Microsoft Win32 Software Development Kit (SDK).

Values	Address of a callback function within your application source code. For example, if your function is declared with the name MyReadCallback, you would pass MyReadCallback as the Callback parameter.

Example

C

Create a notification for the ReadAvail state. Use a timeout of 10 seconds.

```
NCTYPE UINT8 DataBuffer[20];
NCTYPE_STATE _NCFUNC_ MyReadCallback (
            NCTYPE_OBJH ObjHandle,
            NCTYPE STATE CurrentState,
            NCTYPE STATUS Status,
            NCTYPE_ANY_P RefData) {
 if (Status == DnetSuccess) {
    Status = ncReadDnetIO(ObjHandle>, 20, RefData);
 return(NC_ST_READ_AVAIL);
}
void main() {
 NCTYPE STATUS
                    status;
 NCTYPE_OBJH objh;
 status = ncCreateNotification(objh, NC ST READ AVAIL,
        10000, DataBuffer, MyReadCallback);
```

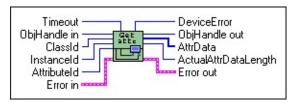
ncGetDnetAttribute (Get DeviceNet Attribute)

Purpose

Get an attribute value from a DeviceNet device using an Explicit Messaging Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncGetDnetAttribute(ObjHandle, NCTYPE_OBJH ClassId, NCTYPE_UINT16 NCTYPE_UINT16 InstanceId, AttributeId, NCTYPE_UINT8 Timeout, NCTYPE_DURATION SizeofAttrData, NCTYPE_UINT16 NCTYPE_ANY_P AttrData, NCTYPE_UINT16_P ActualAttrDataLength NCTYPE_UINT16_P DeviceError);

Input

Object handle of an open Explicit Messaging Object

ClassId Identifies the class which contains the attribute

InstanceId Identifies the instance which contains the attribute

AttributeId Identifies the attribute to get

Timeout Maximum time to wait for response from device

SizeofAttrData Size of AttrData buffer in bytes (C only)

Output

Attribute value received from device

ActualAttrDataLength Actual number of attribute data bytes returned

DeviceError Error codes from device error response

Function Description

ncGetDnetAttribute gets the value of an attribute from a DeviceNet device using an Explicit Messaging Object.

ncGetDnetAttribute executes the Get Attribute Single service on a remote DeviceNet device.

The format of the data returned in AttrData is defined by the DeviceNet data type in the attribute's description. When using LabVIEW, the ncConvertFromDnetRead function can convert this DeviceNet data type into an appropriate LabVIEW data type. When using C, AttrData can point to a variable of the appropriate data type as specified in NI-DNET Data Types.

Parameter Descriptions

ObjHandle

	ObjHandle must contain an object handle returned from the ncOpenDnetExplMsg function. In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

ClassId

Description	Identifies the class which contains the attribute. For descriptions and identifiers for each standard DeviceNet class, refer to the <i>DeviceNet Specification</i> (Volume 2, Chapter 6, <i>The DeviceNet Object Library</i>). Vendor-specific classes are documented by the device vendor. Although the <i>DeviceNet Specification</i> allows 16-bit class IDs, most class IDs are 8-bit. NI-DNET automatically uses the class ID size (16-bit or 8-bit) that is appropriate for your device.
Values	00 to FFFF hex

InstanceId

Description	Identifies the instance which contains the attribute. Instance ID 0 is used to get an attribute from the class itself. Other instance IDs typically are numbered starting at 1. For example, the primary Identity Object in a device uses instance ID 1. Although the <i>DeviceNet Specification</i> allows 16-bit instance IDs, most instance IDs are 8-bit. NI-DNET automatically uses the instance ID size (16-bit or 8-bit) that is appropriate for your device.
Values	00 to FFFF hex

AttributeId

·	Identifies the attribute to get. Attribute IDs are listed in the class and instance descriptions in the <i>DeviceNet Specification</i> . The attribute's description also lists the
	DeviceNet data type for the attribute's value.
Values	00 to FF hex

Timeout

Description	Maximum time to wait for response from device. To get the attribute from the device, an explicit message request for the Get Attribute Single service is sent to the device. After sending the service request, this function must wait for the explicit message response for Get Attribute Single. Timeout specifies the maximum number of milliseconds to wait for the response before giving up. If the timeout expires before the response is received, this function returns a status of BFF62001 hex (CanErrFunctionTimeout).
	appropriate. The special timeout value of FFFFFFF hex is used to wait indefinitely.
Values	1 to 1000
	or
	FFFFFFFF hex (infinite duration, constant
	NC_DURATION_INFINITE)

SizeofAttrData

Description	For C, this is the size of the buffer referenced by AttrData. It is used to verify that you have enough bytes available to store the attribute data. This size is normally obtained using the C language sizeof function and has no direct relation to the number of bytes received on the network.
	For LabVIEW, since the buffer for AttrData is allocated automatically by NI-DNET, this size is not needed.
	The number of bytes allocated for AttrData should be large enough to hold the maximum number of data bytes defined for the attribute.
Values	sizeof (buffer referenced by AttrData)

AttrData

Description Attribute value received from device.	Description
---	-------------

The format of the data returned in AttrData is defined by the DeviceNet data type in the attribute's description. When using LabVIEW, the ncConvertFromDnetRead function can convert this DeviceNet data type into an appropriate LabVIEW data type. When using C, AttrData can point to a variable of the appropriate data type as specified in NI-DNET Data Types.

The number of attribute data bytes returned is the smaller of SizeofAttrData and ActualAttrDataLength.

Values

Attribute data bytes

ActualAttrDataLength

Description	Actual number of attribute data bytes returned. This length
	is obtained from the actual Get Attribute Single response
	message. If this length is greater than SizeofAttrData, only
	SizeofAttrData bytes are returned in AttrData. If this length is
	less than or equal to SizeofAttrData, ActualAttrDataLength
	bytes are valid in AttrData.
Values	0 to 240

DeviceError

Description	Error	codes	from	device's	error	response.
-------------	-------	-------	------	----------	-------	-----------

If the remote device responds successfully to the Get Attribute Single service, the return status is 0 (DnetSuccess), and DeviceError returns 0.

If the remote device returns an error response for the Get Attribute Single service, the return status is BFF62014 hex (DnetErrErrorResponse), and DeviceError returns the error codes from the response.

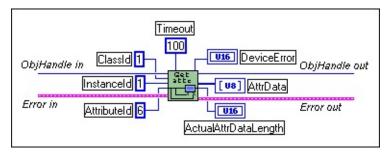
The General Error Code from the device's error response is returned in the low byte of DeviceError. Common values for General Error Code include Attribute Not Supported (14 hex), Object Does Not Exist (16 hex), and Invalid Attribute

	Value (09 hex).
	The Additional Code from the device's error response is returned in the high byte of DeviceError. The Additional Code provides additional information that further describes the error. If no additional information is needed, the value FF hex is placed into this field.
	Values for the General Error Code and Additional Code are documented in the <i>DeviceNet Specification</i> . Common error code values are found in Appendix H, <i>DeviceNet Error Codes</i> , in the <i>DeviceNet Specification</i> . Object-specific error codes are listed in the object description. Vendor-specific error codes are listed in your device's documentation.
Values	Error codes from the device's error response.

Examples

LabVIEW

Get the Serial Number attribute using an Explicit Messaging Object. The Serial Number is contained in the Identity Object (class ID 1, instance ID 1, attribute ID 6). The DeviceNet data type for Device Type is UDINT, for which the LabVIEW data type U32 should be used. The Timeout is 100 ms.



C

Get the Device Type attribute using the Explicit Messaging Object referenced by objh. The Device Type is contained in the Identity Object (class ID 1, instance ID 1, attribute ID 2). The DeviceNet data type for Device Type is UINT, for which the NI-DNET data type NCTYPE_UINT16 should be used.

```
NCTYPE_STATUS status;

NCTYPE_OBJH objh;

NCTYPE_UINT16 device_type;

NCTYPE_UINT16 actual_length;

NCTYPE_UINT16 device_error;

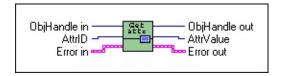
status = ncGetDnetAttribute(objh, 0x01, 0x01, 0x02, 100, sizeof(device_type), &device_type, &actual_length, &device_error);
```

ncGetDriverAttr (Get Driver Attribute) Purpose

Get the value of an attribute in the NI-DNET driver.

Format

LabVIEW



C

NCTYPE_STATUS

ncGetDriverAttr (NCTYPE_OBJH ObjHandle, NCTYPE_ATTRID AttrId, NCTYPE_UINT32 SizeofAttr, NCTYPE_ANY_P Attr)

Input

ObjHandle Object handle of an open Explicit Messaging Object, I/O

Object, or Interface Object

AttrId Identifier of the attribute to get

SizeofAttr Size of the Attr buffer in bytes (C only)

Output

Attr Returned attribute value

Function Description

ncGetDriverAttr gets the value of an attribute in the NI-DNET driver software. Within NI-DNET objects, attributes represent configuration settings, status, and other information.

Since you only need to access NI-DNET driver attributes under special circumstances, ncGetDriverAttr is seldom used. For information about the attributes of each NI-DNET object, refer to NI-DNET Objects.

ncGetDriverAttr only applies to the NI-DNET software on your computer and cannot be used to get an attribute from a remote DeviceNet device. To get an attribute from a remote DeviceNet device, use the ncGetDnetAttribute function.

Parameter Descriptions

ObjHandle

	ObjHandle must contain an object handle returned from ncOpenDnetExplMsg, ncOpenDnetIntf, or ncOpenDnetIO.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

Attrld

•	Identifier of the NI-DNET attribute. Supported attribute identifiers for each NI-DNET object are listed in NI-DNET Objects.
	80000000 to 8000FFFF hex (high bit differentiates from DeviceNet IDs)

SizeofAttr

	For C, this is the size of the buffer referenced by Attr. It is used to verify that you have enough bytes available to store the attribute's value. This size is normally obtained using the C language sizeof function.
	For LabVIEW, since the buffer for Attr is allocated automatically by NI-DNET, this size is not needed.
Values	sizeof (buffer referenced by Attr)

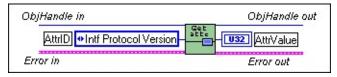
Attr

'	Returned attribute value. The value is usually returned in an unsigned 32-bit integer (and thus Attr is of type NCTYPE_UINT32_P).
Values	Value of NI-DNET attribute

Examples

LabVIEW

Get the DeviceNet protocol version supported by NI-DNET.



C

Get the version of the NI-DNET software using the Interface Object referenced by objh.

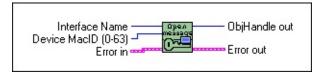
ncOpenDnetExplMsg (Open DeviceNet Explicit Messaging)

Purpose

Configure and open an NI-DNET Explicit Messaging Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncOpenDnetExplMsg(NCTYPE_STRING IntfName, NCTYPE_UINT32 DeviceMacId, NCTYPE_OBJH_P ObjHandle);

Input

IntfName Name of DeviceNet interface

DeviceMacId MAC ID of the remote device

Output

Object handle you use with all subsequent function calls for the Explicit Messaging Object ObjHandle

Function Description

ncOpenDnetExplMsg configures and opens an NI-DNET Explicit Messaging Object and returns a handle that you use with all subsequent function calls for that object.

The Explicit Messaging Object represents an explicit messaging connection to a remote DeviceNet device. Since only one explicit messaging connection is created for a given device, the Explicit Messaging Object is also used for features which apply to the device as a whole.

Use the Explicit Messaging Object to do the following:

- Execute the DeviceNet Get Attribute Single service on the remote device (ncGetDnetAttribute).
- Execute the DeviceNet Set Attribute Single service on the remote device (ncSetDnetAttribute).
- Send any other explicit message request to the remote device and receive the associated explicit message response (ncWriteDnetExplMsg, ncReadDnetExplMsg).
- Configure NI-DNET settings that apply to the entire remote device.

Parameter Descriptions

IntfName

•	Name of the DeviceNet interface as an ASCII string with format "DNET x ", where x is a decimal number starting at zero that indicates which DeviceNet interface is being used. You associate DeviceNet interface names with physical ports using Measurement & Automation Explorer (MAX).
Values	"DNET0", "DNET1","DNET31" In LabVIEW, the interface name is selected from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

DeviceMacId

Description	MAC ID (device address) of the remote DeviceNet device.
	Many devices use physical switches to set their MAC ID. For such devices, you can usually determine the device's MAC ID by examining those switches. MAC ID 63 is usually reserved for new devices (many devices use 63 as the factory default).
	If you do not know the MAC ID of your DeviceNet device, NI-DNET provides a utility which can display the MAC ID for you. This utility, Configurator, is described in the NI-DNET User Manual.
Values	0 to 63

ObjHandle

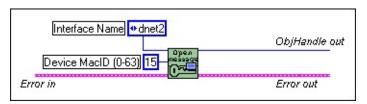
If the ncOpenDnetExplMsg function is successful, a handle to the newly opened Explicit Messaging Object is returned in ObjHandle. This handle is used with all subsequent function calls for that Explicit Messaging Object.
The functions most commonly used with the Explicit Messaging Object are ncGetDnetAttribute and ncSetDnetAttribute.

Values The encoding of ObjHandle is internal to NI-DNET.

Examples

LabVIEW

Open an Explicit Messaging Object using interface "DNET2" and device MAC ID $15.\,$



C

Open an Explicit Messaging Object using interface "DNET0" and device MAC ID $12.\,$

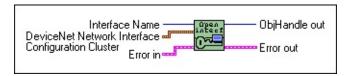
```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
status = ncOpenDnetExplMsg("DNET0", 12, &objh);
```

ncOpenDnetIntf (Open DeviceNet Interface) Purpose

Configure and open an NI-DNET Interface Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncOpenDnetIntf(NCTYPE_STRINGIntfName, NCTYPE_UINT32 IntfMacId, NCTYPE_UINT32> BaudRate, NCTYPE_UINT32 PollMode, NCTYPE_OBJH_PObjHandle);

Input

IntfName Name of DeviceNet interface

IntfMacId MAC ID of the DeviceNet interface

BaudRate Baud rate

PollMode Communication scheme for all polled I/O connections

Output

Object handle you use with all subsequent function calls for the Interface Object ObjHandle

Function Description

ncOpenDnetIntf configures and opens an NI-DNET Interface Object and returns a handle that you use with all subsequent function calls for that object.

The Interface Object represents a DeviceNet interface. Since this interface acts as a device on the DeviceNet network much like any other device, it is configured with its own MAC ID and baud rate.

Use the Interface Object to do the following:

- Configure NI-DNET settings which apply to the entire interface.
- Start and stop communication for all NI-DNET objects associated with the interface.

The Interface Object must be the first NI-DNET object opened by your application, and thus ncOpenDnetIntf must be the first NI-DNET function called by your application.

Parameter Descriptions

IntfName

Description	Name of the DeviceNet interface as an ASCII string with format "DNETx", where x is a decimal number starting at zero that indicates which DeviceNet interface is being used. You associate DeviceNet interface names with physical ports using Measurement & Automation Explorer (MAX).
Values	"DNET0", "DNET1","DNET31" In LabVIEW, the interface name is selected from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

IntfMacId

	MAC ID (device address) of the DeviceNet interface. This is the MAC ID used by your DeviceNet interface for communication with other DeviceNet devices.
	A device's MAC ID indicates the priority of its DeviceNet messages on the network, with lower numbered MAC IDs having higher priority. If your DeviceNet interface is the only master in the network (the usual case), this MAC ID is often set to 0.
Values	0 to 63

BaudRate

Description	Baud rate used for communication on the network connected to the DeviceNet interface. The DeviceNet protocol supports baud rates of 125,000, 250,000, and 500,000 b/s.
	125000, 250000, or 500000 In LabVIEW, you select the baud rate from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

PollMode

Description Determines the communication scheme used for all polled I/O connections in which the interface acts as a master. The poll mode determines the overall scheme used to transmit poll requests to slave devices.

Automatic

The default poll mode is Automatic. Use this mode if you do not want to specify exact timing for polled and strobed I/O connections. In Automatic mode, the NI-DNET software automatically calculates a safe rate for production of all poll requests and strobe requests. This mode is similar to Scanned mode, except that you do not need to specify a valid ExpPacketRate for each polled/strobed I/O Object (ExpPacketRate is ignored).

If you use Automatic, you cannot call the ncOpenDnetIO function while communicating (after ncOperateDnetIntf with Start), because the automatic rate calculation occurs during Start. Use Scanned or Individual if you need to open I/O connections while communicating.

Scanned

This mode enables the traditional scanned I/O scheme for polled and strobed I/O connections. In Scanned mode, all poll requests and strobe requests are produced in quick succession, then NI-DNET waits to receive individual responses. The benefits of scanned I/O are reduced overhead and improved overall determinism on the DeviceNet network.

When using Scanned mode, since all poll and strobe requests are produced at the same time, you normally set the ExpPacketRate for all polled and strobed I/O Objects to a common value.

If you need to isolate devices that are slow to respond to poll requests, it is possible to use different ExpPacketRate values while still maintaining the benefits of scanned I/O. You can set all ExpPacketRate values for polled I/O Objects as two groups: one foreground group, and a second background group whose ExpPacketRate is an exact multiple of the foreground group's. All strobed I/O must use the same rate as the foreground group for polled I/O. For example, you can set some polled I/O (and all strobed I/O) to a common foreground rate of 100 ms, and other polled I/O to a background rate of 500 ms. To maintain overall network determinism, the background poll requests are interspersed evenly among each foreground scan.

Individual

This mode enables you to configure poll rates individually for each polled I/O connection. In Individual mode, poll requests are not produced as a group, but instead each polled I/O connection communicates at an independent rate. The rate at which each poll request is produced is determined solely by the ExpPacketRate of that connection's I/O Object.

Use individual polling when you have detailed knowledge of the time it takes each device to perform its physical measurement or control function. For example, if you have a discrete input device capable of acquiring a new measurement every 10 ms, an analog input device with a measurement rate of 45 ms, and a temperature sensor with a measurement rate of 200 ms, you could use individual polling to communicate with each device at its exact measurement rate. Since communication occurs only at the actual rate needed for each device, individual polling often provides optimum network usage.

For additional information on PollMode and ExpPacketRate, refer to the *NI-DNET User Manual*.

Values

Automatic (constant NC_POLL_AUTO, value 0)

Scanned (constant NC POLL SCAN, value 1)

Individual (constant NC_POLL_INDIV, value 2)

In LabVIEW, you select the poll mode from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

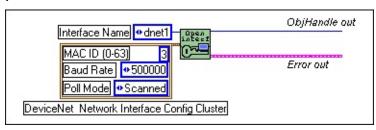
ObjHandle

	If the ncOpenDnetIntf function is successful, a handle to the newly opened Interface Object is returned in ObjHandle. This handle is used with all subsequent function calls for that Interface Object.
	The function most commonly used with the Interface Object is ncOperateDnetIntf.
Values	The encoding of ObjHandle is internal to NI-DNET.

Examples

LabVIEW

Open Interface Object "DNET1" using baud rate 500000, MAC ID 3, and poll mode Scanned.



C

Open Interface Object "DNET0" using baud rate 125000, MAC ID 0, and poll mode Automatic.

```
NCTYPE_STATUS status;

NCTYPE_OBJH objh;

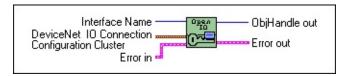
status = ncOpenDnetIntf("DNET0", 0, 125000, NC_POLL_AUTO, &objh);
```

ncOpenDnetIO (Open DeviceNet I/O) Purpose

Configure and open an NI-DNET I/O Object.

Format

LabVIEW



C

NCTYPE_STATUS

Input

IntfName Name of DeviceNet interface

DeviceMacId MAC ID of the remote device

ConnectionType Type of I/O connection

InputLength Number of input bytes

OutputLength Number of output bytes

ExpPacketRate Expected rate of I/O message (packet) production

Output

Object handle you use with all subsequent function calls for the I/O Object ObjHandle

Function Description

ncOpenDnetIO configures and opens an NI-DNET I/O Object and returns a handle that you use with all subsequent function calls for that object.

The I/O Object represents an I/O connection to a remote DeviceNet device. The I/O Object usually represents I/O communication as a master with a remote slave device. If your computer is essentially being used as the primary controller of your DeviceNet devices, you should configure I/O communication as a master.

You can also configure the I/O Object for I/O communication as a slave with a remote master. If your computer is essentially being used as a peripheral device for another primary controller, you can configure I/O communication as a slave. This is done by setting the I/O Object's DeviceMacId to the same MAC ID as the Interface Object (IntfMacId parameter of ncOpenDnetIntf).

The I/O Object supports as many master/slave I/O connections as currently allowed by the *DeviceNet Specification* (version 2.0). This means that you can use polled, strobed, and COS/cyclic I/O connections simultaneously for a given device. As specified by the *DeviceNet Specification*, you can only use one master/slave I/O connection of a given type for each device (MAC ID). For example, you cannot open two polled I/O connections for the same device.

Use the I/O Object to do the following:

- Read data from the most recent message received on the I/O connection (ncReadDnetIO).
- Write data for the next message produced on the I/O connection (ncWriteDnetIO).

Parameter Descriptions

IntfName

•	Name of the DeviceNet interface as an ASCII string with format "DNET x ", where x is a decimal number starting at zero that indicates which DeviceNet interface is being used. You associate DeviceNet interface names with physical ports using Measurement & Automation Explorer (MAX).
	"DNET0", "DNET1","DNET31" In LabVIEW, you select the interface name from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

DeviceMacId

Description MAC ID (device address) of the remote DeviceNet device. Many devices use physical switches to set their MAC ID. For such devices, you can usually determine the device's MAC ID by examining those switches. MAC ID 63 is usually reserved for new devices (many devices use 63 as the factory default). If you do not know the MAC ID of your DeviceNet device, NI-DNET provides a utility which can display the MAC ID for you. This utility, Configurator, is described in the *NI-DNET* User Manual. For I/O communication as a master to a remote slave device (the usual case), DeviceMacId is the MAC ID of the remote DeviceNet slave device, and thus must be different than the MAC ID of your DeviceNet interface. If you want to configure I/O communication as a slave with a remote master, set DeviceMacId to the same MAC ID as your DeviceNet interface (the IntfMacId parameter of your previous call to ncOpenDnetIntf). By associating the I/O Object with your

DeviceNet interface in this manner, you indicate that it

represents I/O communication as a slave.

ConnectionType

Description Type of master/slave I/O connection. The connection type is either Polled, Strobed, change-of-state (COS), or Cyclic. As specified by the *DeviceNet Specification*, you can use only one master/slave I/O connection of a given type for each device (MAC ID). For example, you cannot open two polled I/O connections for the same device.

> If you do not know the I/O connection types supported by your DeviceNet device, NI-DNET provides a utility which queries the device for both this information and the device's supported input and output lengths. This utility, Configurator, is described in the *NI-DNET User Manual*. Change-of-state (COS) and cyclic I/O connections are acknowledged by default. If you want to suppress acknowledgments for these I/O connections, set the Ack Suppress driver attribute to true prior to starting communication. For more information, refer to the description of the **I/O** Object.

Values

Poll (constant NC CONN POLL, value 0)

Strobe (constant NC_CONN_STROBE, value 1)

COS (constant NC_CONN_COS, value 2)

Cyclic (constant NC_CONN_CYCLIC, value 3)

In LabVIEW, you select the connection type from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

InputLength

Description Number of input bytes for the I/O connection. This is the number of bytes read from the I/O connection using the ncReadDnetIO function.

> The following information is specific to the ConnectionType setting.

Poll, COS, and Cyclic For these I/O connection types, the input length is the same as the number of bytes consumed from the remote device. Strobe as master (DeviceMacId not equal to IntfMacId) For this I/O connection, the input length is the same as the number of bytes consumed from the strobe response message, and must have a value from 0 to 8. Strobe as slave (DeviceMacId equal to IntfMacId) For this I/O connection, the input length must have a value of 1. The input data consists of a single Boolean value (bit) obtained from the master's strobe command message using IntfMacId. This Boolean value is returned from the ncReadDnetIO function as a single byte. Values Poll, COS, and Cyclic: 0 to 255 Strobe as master (DeviceMacId not equal to IntfMacId): 0 to 8 Strobe as slave (DeviceMacId equal to IntfMacId): 1

OutputLength

Description	Number of output bytes for the I/O connection. This is the number of bytes written to the I/O connection using the ncWriteDnetIO function.
	The following information is specific to the ConnectionType setting.
	Poll, COS, and Cyclic
	For these I/O connections types, the output length is the same as the number of bytes produced to the remote device.
	Strobe as master (DeviceMacId not equal to IntfMacId)
	For this I/O connection, the output length must have a value of 1. The output data consists of a single Boolean value (bit) which is placed into the strobe command message using

	DeviceMacId. This Boolean value is provided to the ncWriteDnetIO function as a single byte.
	Strobe as slave (DeviceMacId equal to IntfMacId)
	For this I/O connection, the output length must have a value from 0 to 8. The output length is the same as the number of bytes produced in the strobe response message.
Values	Poll, COS, and Cyclic: 0 to 255
	Strobe as master (DeviceMacId not equal to IntfMacId): 1
	Strobe as slave (DeviceMacId equal to IntfMacId): 0 to 8

ExpPacketRate

Description	Expected rate of I/O message (packet) production in
	milliseconds.

As specified in the *DeviceNet Specification*, the expected packet rate is used to trigger data productions. The expected packet rate is also used for the watchdog timer to verify that the device on the other side of the I/O connection still exists and is producing data as expected. The expected packet rate of each I/O connection is a major factor in determining the overall performance of your DeviceNet network.

The following information is specific to the ConnectionType setting and the PollMode setting of your Interface Object.

Strobe with Automatic poll mode

When using the Automatic poll mode, the ExpPacketRate setting is ignored for strobed I/O Objects. The rate of production for the strobe command message is determined automatically by NI-DNET.

Strobe with Scanned or Individual poll mode

When using the Scanned or Individual poll mode, you must set the ExpPacketRate to the same value for all strobed I/O Objects. Since a single strobe command message is

produced for all strobed I/O connections, the rate of production for that message must be identical for all strobed I/O Objects.

Poll with Automatic poll mode

When using the Automatic poll mode, the ExpPacketRate setting is ignored for polled I/O Objects. NI-DNET automatically determines the rate of production for the poll command messages.

Poll with Scanned poll mode

When using the Scanned poll mode, since all poll and strobe requests are produced at the same time, you normally set the ExpPacketRate for all polled/strobed I/O Objects to a common value.

If you need to isolate devices that are slow to respond to poll requests, it is possible to use different ExpPacketRate values while still maintaining the benefits of scanned I/O. You can set all ExpPacketRate values for polled I/O Objects as two groups, one foreground group, and a second background group whose ExpPacketRate is an exact multiple of the foreground group's. All strobed I/O must use the same rate as the foreground group for polled I/O. For example, you can set some polled I/O (and all strobed I/O) to a common foreground rate of 100 ms, and other polled I/O to a background rate of 500 ms. To maintain overall network determinism, the background poll requests are interspersed evenly among each foreground scan.

Poll with Individual poll mode

When using the Individual poll mode, the ExpPacketRate determines the rate at which the poll request of each polled I/O Object is produced. Although all strobed I/O Objects must still use the same rate, each polled I/O Object communicates at a rate which is independent of all other I/O connections.

Change-of-state (COS) with any poll mode

For COS I/O Objects, the ExpPacketRate is used solely to verify that the I/O connection still exists. If no change in data

produces I/O message within the expected packet rate, the previous data is produced again to maintain the I/O connection. Since this rate is used solely to maintain the I/O connection, it is often set to a large value, such as 10000 (10 seconds).

In addition to the expected packet rate, COS I/O connections also produce an I/O message when a change is detected in the data. These I/O change messages do not occur at a predetermined rate. The time between each I/O change message depends on when an actual change takes place and how fast the device can measure new data and detect changes.

Cyclic with any poll mode

For cyclic I/O Objects, the ExpPacketRate determines the rate at which the I/O message is produced. Each cyclic I/O Object communicates at a rate which is independent of all other I/O connections.

Note regarding I/O as a slave (DeviceMacId equal to IntfMacId)

The ExpPacketRate setting applies only to I/O Objects used for communication as a master (the usual case). For I/O Objects used for communication as a slave, this setting is ignored because the remote master determines the expected packet rate on behalf of your slave I/O connection.

Values

1 to 60000

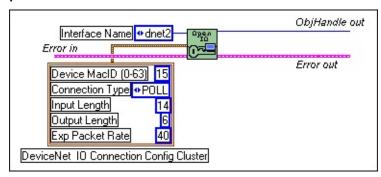
ObjHandle

	If the ncOpenDnetIO function is successful, a handle to the newly opened I/O Object is returned in ObjHandle. This handle is used with all subsequent function calls for that I/O Object.
	The functions most commonly used with the I/O Object are ncReadDnetIO and ncWriteDnetIO.
Values	The encoding of ObjHandle is internal to NI-DNET.

Examples

LabVIEW

Open an I/O Object using interface "DNET2", device MAC ID 15, connection type Poll, input length 14, output length 6, and expected packet rate 40 ms.



C

Open an I/O Object using interface "DNET0", device MAC ID 12, connection type Strobe, input length 2, output length 1, and expected packet rate 100 ms.

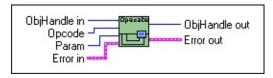
NCTYPE_STATUS status; NCTYPE_OBJH objh; status = ncOpenDnetIO("DNET0", 12, ,NC_CONN_STROBE, 2, 1, 100, &objh)

ncOperateDnetIntf (Operate DeviceNet Interface) Purpose

Perform an operation on an NI-DNET Interface Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncOperateDnetIntf (NCTYPE_OBJH ObjHandle, NCTYPE_UINT32 Opcode, NCTYPE_UINT32 Param);

Input

ObjHandle Object handle of an open Interface Object

Opcode Operation code indicating which operation to perform

Param Parameter whose meaning is defined by Opcode

Output

None

Function Description

ncOperateDnetIntf operates on an NI-DNET Interface Object.

Use ncOperateDnetIntf to start and stop all DeviceNet communication for the associated interface, including all explicit messaging and I/O connections. After you open the Explicit Messaging Objects and I/O Objects required by your application, you must use ncOperateDnetIntf to start communication. You must also use ncOperateDnetIntf to stop communication before terminating your application.

Parameter Descriptions

ObjHandle

	ObjHandle must contain an object handle returned from the ncOpenDnetIntf function.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the Interface Object.
Values	The encoding of ObjHandle is internal to NI-DNET.

Opcode

Description	Determines which operation to perform on the Interface
	Object.

Start

Start all DeviceNet communication for the associated interface. For each Explicit Messaging Object and I/O Object which has been opened for the interface (same IntfName), this operation establishes the DeviceNet connection with the remote device. When the operation establishes I/O connections, it places outputs into active mode (data is produced on the network). If the default output data (all bytes zero) is not valid for your application, use ncWriteDnetIO for each I/O Object to initialize valid output data prior to starting communication. If the interface has already been started, this operation has no effect.

Stop all DeviceNet communication for the associated interface. For each Explicit Messaging Object and I/O Object which has been opened for the interface, this operation closes the DeviceNet connection with the remote device. Although closing all NI-DNET objects implicitly stops communication, you should perform this operation prior to calling ncCloseObject. If the interface has already been stopped, this operation has no effect.

Active

Place the outputs of all I/O connections into active mode. When an I/O connection is in active mode, it produces data in its outgoing I/O message. This operation is used after a previous Idle to restore normal communication on all I/O Objects associated with the interface. If the interface has already been placed into active mode or is stopped, this operation has no effect.

Idle

Place the outputs of all I/O connections into the idle mode. When an I/O connection is in the idle mode, it does not produce data in its outgoing I/O message, but the I/O connection is kept open by producing an I/O message with zero data bytes. Use this operation when valid output data is no longer available from your application, such as when a control algorithm has been paused. If the interface has already been placed into idle mode or is stopped, this operation has no effect.



Note The *DeviceNet Specification* does not clearly define the behavior of a slave device on reception of an idle (zero length) I/O message. Many slave devices exhibit unexpected behavior when the Idle operation is used. If you need to suspend your application, but want to keep I/O connections open, you should provide valid idle values for outputs using ncWriteDnetIO rather than use the Idle operation.

Values

Start (constant NC_OP_START, value 1)

Stop (constant NC_OP_STOP, value 2)

Active (constant NC_OP_ACTIVE, value 4)

Idle (constant NC_OP_IDLE, value 5)

In LabVIEW, you select the operation code from an enumerated list. The LabWindows/CVI function panel also provides an enumerated list.

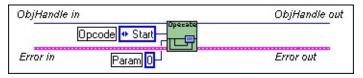
Param

·	The meaning of Param is defined by each operation code (Opcode). Since none of the operations currently use this additional parameter, it is ignored and you should normally set it to zero. In the future, if new operations require some form of qualifying information, this parameter might be used.
Values	0

Examples

LabVIEW

Start communication using an Interface Object.



C

Stop communication for the Interface Object referenced by objh.

```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
status = ncOperateDnetIntf(objh, NC_OP_STOP, 0);
```

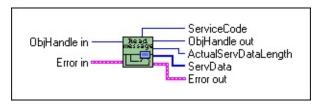
ncReadDnetExplMsg (Read DeviceNet Explicit Message)

Purpose

Read an explicit message response from an Explicit Messaging Object.

Format

LabVIEW



C

NCTYPE_STATUS

Input

ObjHandle Object handle of an open Explicit Messaging Object SizeofServData Size of ServData buffer in bytes (C only)

Output

ServiceCode DeviceNet service code from response

ServData Service data from response

ActualServDataLength Actual number of service data bytes in response

Function Description

ncReadDnetExplMsg reads an explicit message response from an Explicit Messaging Object.

The two most commonly used DeviceNet explicit messages are the Get Attribute Single service and the Set Attribute Single service. The easiest way to execute the Get Attribute Single service on a remote device is to use the NI-DNET ncGetDnetAttribute function. The easiest way to execute the Set Attribute Single service on a remote device is to use the NI-DNET ncSetDnetAttribute function.

To execute services other than Get Attribute Single and Set Attribute Single, use the following sequence of function calls: ncWriteDnetExplMsg, ncWaitForState, ncReadDnetExplMsg. The ncWriteDnetExplMsg function sends an explicit message request to a remote DeviceNet device. The ncWaitForState function waits for the explicit message response, and the ncReadDnetExplMsg function reads that response.

Some of the DeviceNet services which use ncReadDnetExplMsg are Reset, Save, Restore, Get Attributes All, and Set Attributes All. Although the *DeviceNet Specification* defines the overall format of these services, in most cases their meaning and service data are object specific or vendor specific. Unless your device requires such services and documents them in detail, you probably do not need them for your application. For more information, refer to the *NI-DNET User Manual*.

Parameter Descriptions

ObjHandle

Description	ObjHandle must contain an object handle returned from ncOpenDnetExplMsg.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

ServiceCode

Description	Identifies the service response as either success or error. If the response is success, this value is the same as the ServiceCode of the request (ncWriteDnetExplMsg), and the ServData bytes are formatted as defined by the service. If the response is error, this value is 14 hex, ServData[0] contains a General Error Code, and ServData[1] contains an Additional Code. Either the <i>DeviceNet Specification</i> or the object itself define the error codes.
	Although the <i>DeviceNet Specification</i> requires the high bit of the service code (hex 80) to be set in all explicit message responses, NI-DNET clears this response indicator so that you can compare the actual service code to the value used with ncWriteDnetExplMsg.
Values	Same as the ServiceCode of ncWriteDnetExplMsg (success response) or 14 hex (error response)

SizeofServData

·	For C, this is the size of the buffer referenced by ServData. Use it to verify that you have enough bytes available to store the service data from the response. This size is normally obtained using the C language sizeof function and has no direct relation to the number of bytes received on the network.
	For LabVIEW, since the buffer for ServData is allocated

	automatically by NI-DNET, this size is not needed.
	The number of bytes allocated for ServData should be large enough to hold the maximum number of service data response bytes defined for the service.
Values	sizeof (buffer referenced by ServData)

ServData

Description	Service data bytes from response. If the response is success, these bytes are formatted as defined by the service. If the response is error, the first byte (ServData[0]) contains a General Error Code, and the second byte (ServData[1]) contains an Additional Code. Either the DeviceNet Specification or the object itself define the error codes.
	The number of service data bytes returned is the smaller of SizeofServData and ActualServDataLength.
Values	Service data bytes from response

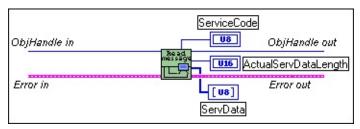
ActualServDataLength

·	Actual number of service data bytes in response. This length is obtained from the actual response message. If this length is greater than SizeofServData, only SizeofServData bytes are returned in ServData. If this length is less than or equal to SizeofServData, ActualServDataLength bytes are valid in ServData.
Values	0 to 240

Examples

LabVIEW

Read an explicit message response from an Explicit Messaging Object.



C

Read an explicit message response from the Explicit Messaging Object referenced by objh.

```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
NCTYPE_UINT8 servcode;
NCTYPE_UINT8 servdata[20];
NCTYPE_UINT16 actual_len;
status = ncReadDnetExplMsg(objh, &servcode, 20, servdata, &actual_len);
```

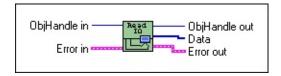
ncReadDnetIO (Read DeviceNet I/O)

Purpose

Read input data from an I/O Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncReadDnetIO(NCTYPE_OBJH ObjHandle, NCTYPE_UINT32 SizeofData, NCTYPE_ANY_P Data);

Input

ObjHandle Object handle of an open I/O Object

SizeofData Size of Data buffer in bytes (C only)

Output

Data Input data

Function Description

ncReadDnetIO reads input data from an NI-DNET I/O Object.

Since each I/O Object continuously acquires input data from the DeviceNet network, you normally wait for new input to become available prior to calling ncReadDnetIO. By waiting for new input data, your application can handle I/O data at the same rate as the DeviceNet I/O communication. You can use the function ncCreateNotification (C only) or ncWaitForState (C or LabVIEW) to wait for new input data.

ncReadDnetIO normally returns input data bytes obtained from the input assembly of a remote DeviceNet slave device. The format of this input assembly is normally documented either by the device vendor or within the *DeviceNet Specification* itself.

The bytes of a device's input assembly often consist of multiple data members rather than a single value. For C, you can often obtain each data member from the input bytes by using typecasting. For LabVIEW, you can often obtain each data member from the input bytes using the ncConvertFromDnetRead function. For more information on input assemblies and how to obtain individual data members, refer to the *NI-DNET User Manual*.

Parameter Descriptions

ObjHandle

Description	ObjHandle must contain an object handle returned from ncOpenDnetIO.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

SizeofData

	For C, SizeofData is the size of the buffer referenced by Data. Use it to verify that you have enough bytes available to store the input bytes. This size is normally obtained using the C language sizeof function and has no direct relation to the number of bytes received on the network. For LabVIEW, since the buffer for Data is allocated automatically by NI-DNET, this size is not needed. The actual number of bytes received on the I/O connection is determined by the InputLength parameter of ncOpenDnetIO and not this size.
Values	sizeof (buffer referenced by Data)

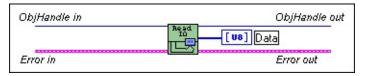
Data

•	Input data. The format of these input bytes is specific to your DeviceNet device.
Values	Input data bytes

Examples

LabVIEW

Read 20 input bytes from an I/O Object.



C

Read 10 input bytes from the I/O Object referenced by objh.

```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
NCTYPE_UINT8 input[10];
status = ncReadDnetIO(objh, 10, input);
```

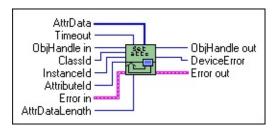
ncSetDnetAttribute (Set DeviceNet Attribute)

Purpose

Set an attribute value for a DeviceNet device using an Explicit Messaging Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncSetDnetAttribute(NCTYPE_OBJH ObjHandle, NCTYPE UINT16 ClassId, NCTYPE_UINT16 InstanceId, AttributeId, NCTYPE_UINT8 Timeout, NCTYPE DURATION NCTYPE UINT16 AttrDataLength, NCTYPE_ANY_P AttrData NCTYPE_UINT16_P DeviceError);

Input

Object handle of an open Explicit Messaging Object

ClassId Identifies the class which contains the attribute

InstanceId Identifies the instance which contains the attribute

AttributeId Identifies the attribute to set

Timeout Maximum time to wait for response from device

AttrDataLength Number of attribute data bytes to set

AttrData Attribute value to set in device

Output

DeviceError Error codes from device's error response

Function Description

ncSetDnetAttribute sets the value of an attribute for a DeviceNet device using an Explicit Messaging Object.

ncSetDnetAttribute executes the Set Attribute Single service on a remote DeviceNet device.

The DeviceNet data type in the attribute's description defines the format of the data provided in AttrData. When using LabVIEW, the ncConvertForDnetWrite function can convert this DeviceNet data type from an appropriate LabVIEW data type. When using C, AttrData can point to a variable of the appropriate data type as specified in NI-DNET Data Types.

Parameter Descriptions

ObjHandle

	ObjHandle must contain an object handle returned from the ncOpenDnetExplMsg function.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

ClassId

Description	Identifies the class which contains the attribute. You can find descriptions and identifiers for each standard DeviceNet class in the <i>DeviceNet Specification</i> (Volume 2, Chapter 6, <i>The DeviceNet Object Library</i>). The device vendor documents vendor-specific classes. Although the <i>DeviceNet Specification</i> allows 16-bit class IDs, most class IDs are 8-bit. NI-DNET automatically used the class ID size (16-bit or 8-bit) that is appropriate for your device.
Values	00 to FFFF hex

InstanceId

	Identifies the instance which contains the attribute. Instance ID 0 sets an attribute in the class itself. Other instance IDs typically are numbered starting at 1. For example, the primary Identity Object in a device uses instance ID 1. Although the <i>DeviceNet Specification</i> allows 16-bit instance IDs, most instance IDs are 8-bit. NI-DNET automatically uses the instance ID size (16-bit or 8-bit) that is appropriate for your device.
Values	00 to FFFF hex

AttributeId

Identifies the attribute to set. The class and instance descriptions list attribute IDs. The attribute's description also
lists the DeviceNet data type for the attribute's value.

Values	00 to FF hex
--------	--------------

Timeout

Maximum time to wait for response from device. To set the attribute in the device, an explicit message request for the Set Attribute Single service is sent to the device. After sending the service request, this function must wait for the explicit message response for Set Attribute Single. Timeout specifies the maximum number of milliseconds to wait for the response before giving up. If the timeout expires before the response is received, this function returns a status of BFF62001 hex (CanErrFunctionTimeout). For most DeviceNet devices, a Timeout of 100 ms is appropriate. The special timeout value of FFFFFFFF hex is used to wait indefinitely.
1 to 1000
or
FFFFFFF hex (infinite duration, constant NC_DURATION_INFINITE)

AttrDataLength

•	Number of attribute data bytes to set. This length also specifies the number of bytes provided in AttrData.
Values	0 to 239

AttrData

Description	Attribute value to set in device.
	The DeviceNet data type in the attribute's description defines the format of the data provided in AttrData. When using LabVIEW, the ncConvertForDnetWrite function can convert this DeviceNet data type from an appropriate LabVIEW data type. When using C, AttrData can point to a

	variable of the appropriate data type as specified in <u>NI-DNET Data Types</u> .
	The AttrDataLength parameter specifies the number of attribute data bytes to set.
Values	Attribute value to set in device.

DeviceError

Description Error codes from device's error response.

If the remote device responds successfully to the Set Attribute Single service, the return status is 0 (DnetSuccess), and DeviceError returns 0.

If the remote device returns an error response for the Set Attribute Single service, the return status is BFF62014 hex (DnetErrErrorResponse), and DeviceError returns the error codes from the response.

The General Error Code from the device's error response is returned in the low byte of DeviceError. Common values for General Error Code include Attribute Not Supported (14 hex), Object Does Not Exist (16 hex), and Invalid Attribute Value (09 hex).

The Additional Code from the device's error response is returned in the high byte of DeviceError. The Additional Code provides additional information that further describes the error. If no additional information is needed, the value FF hex is placed into this field.

The *DeviceNet Specification* documents values for the General Error Code and Additional Code. You can find common error code values in Appendix H, *DeviceNet Error Codes*, in the *DeviceNet Specification*. The object description lists object-specific error codes. Your device's documentation lists vendor-specific error codes.

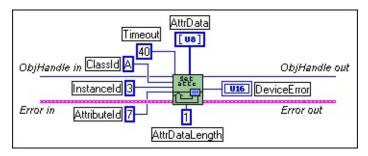
Values

Error codes from the device's error response.

Examples

LabVIEW

Set the Input Range attribute of an Analog Input Object. The Input Range is contained in instance 3 of an Analog Input Object (class ID 0A hex, instance ID 3, attribute ID 7). The DeviceNet data type for Input Range is USINT, for which the LabVIEW data type U8 should be used. The Timeout is 40 ms.



C

Set the MAC ID attribute of a remote DeviceNet device using the Explicit Messaging Object referenced by objh. The MAC ID is contained in the DeviceNet Object (class ID 3, instance ID 1, attribute ID 1). The DeviceNet data type for Device Type is USINT, for which the NI-DNET data type NCTYPE_UINT8 should be used.

```
NCTYPE_STATUS status;

NCTYPE_OBJH objh;

NCTYPE_UINT8 mac_id;

NCTYPE_UINT16 device_error;

mac_id = 12;

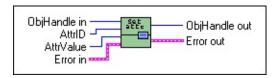
status = ncSetDnetAttribute(objh, 0x03, 0x01, 0x01, 100, 1, &mac_id, &device_e
```

ncSetDriverAttr (Set Driver Attribute) Purpose

Set the value of an attribute in the NI-DNET driver.

Format

LabVIEW



C

NCTYPE_STATUS

ncSetDriverAttr (NCTYPE_OBJH ObjHandle, NCTYPE_ATTRID AttrId, NCTYPE_UINT32 SizeofAttr, NCTYPE_ANY_P Attr)

Input

ObjHandle Object handle of an open Explicit Messaging Object, I/O

Object, or Interface Object

AttrId Identifier of the attribute to set

SizeofAttr Size of the Attr buffer in bytes (C only)

Attr New attribute value

Output

None

Function Description

ncSetDriverAttr sets the value of an attribute in the NI-DNET driver software. NI-DNET objects use attributes to represent configuration settings, status, and other information.

Since you only need to access NI-DNET driver attributes under special circumstances, you seldom need to use ncSetDriverAttr. For information about the attributes of each NI-DNET object, refer to NI-DNET Objects.

ncSetDriverAttr only applies to the NI-DNET software on your computer and cannot be used to set an attribute in a remote DeviceNet device. To set an attribute in a remote DeviceNet device, use ncSetDnetAttribute.

Parameter Descriptions

ObjHandle

Description	ObjHandle must contain an object handle returned from ncOpenDnetExplMsg, ncOpenDnetIntf, or ncOpenDnetIO.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

Attrld

-	Identifier of the NI-DNET attribute. For each NI-DNET object, a list of supported attribute identifiers is provided in NI-DNET Objects.
	80000000 to 8000FFFF hex (high bit differentiates from DeviceNet IDs)

SizeofAttr

-	For C, SizeofAttr is the size of the buffer referenced by Attr. It is used to verify that the Attr buffer is large enough to hold the attribute's new value. This size is normally obtained using the C language sizeof function.
	For LabVIEW, since Attr is obtained directly as an input, this size is not needed.
Values	sizeof (buffer referenced by Attr)

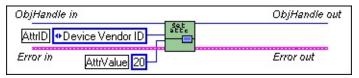
Attr

·	New attribute value. The value is usually provided in an unsigned 32-bit integer (and thus Attr is of type NCTYPE_UINT32_P).
Values	New value of NI-DNET attribute

Examples

LabVIEW

Verify vendor ID 20 for the DeviceNet device referenced by an Explicit Messaging Object.



C

Suppress acknowledgments for the COS I/O Object referenced by objh.

```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
NCTYPE_BOOL ack_sup;
ack_sup = NC_TRUE;
status = ncSetDriverAttr(objh, NC_ATTR_ACK_SUPPRESS, sizeof(ack_sup), 8
```

ncStatusToString (Status To String)

Purpose

Convert status returned from an NI-DNET function into a descriptive string.

Format

LabVIEW

Not applicable

For LabVIEW, NI-DNET functions use the standard error in and error out clusters for status information. You can view error descriptions using built-in LabVIEW features such as **Explain Error** in the **Help** menu, or the **Simple Error Handler** VI in your diagram.

C

void

ncStatustoString(
NCTYPE_STATUS Status,
NCTYPE_UINT32 SizeofString,
NCTYPE_STRING String);

Input

Status Status returned from a previous function call

SizeofString Size of String buffer in bytes

Output

String Textual string which describes the function status

Function Description

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

NI-DNET Status Codes

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

ncStatusToString converts a status value returned from an NI-DNET function into a descriptive string. By displaying this string when an error or warning is detected, you can avoid interpretation of the numeric code to debug the problem.

The ncStatustoString function is not applicable to LabVIEW programming. For LabVIEW, NI-DNET functions use the standard error in and error out clusters for status information. You can view error descriptions using built-in LabVIEW features such as **Explain Error** in the **Help** menu, or the **Simple Error Handler** VI in your diagram.

If you want to avoid displaying error messages while debugging your application, you can use the Explain.exe utility. This console application is in the Utilities subfolder of the NI-DNET installation folder, which is typically \Program Files\National Instruments\NI-DNET\Utilities. You enter an NI-DNET status code in the command line (such as Explain 0xBFF62001), and the utility displays the description.

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

Parameter Descriptions

Status

Status must contain a status value returned from a previous call to an NI-DNET function. You normally call ncStatustoString only when the status is nonzero, indicating an error or warning condition.
Value of data type NCTYPE_STATUS, returned from an NI-DNET function call

SizeofString

Description	SizeofString is the size of the buffer referenced by String. The ncStatustoString function copies at most SizeofString bytes into the string and cuts off the text as needed. You can normally obtain this size using the C language sizeof function. Although you can often obtain an adequate description with fewer bytes, a 512-byte buffer is large enough to hold any NI-DNET status description.
Values	sizeof (buffer referenced by String)

String

	Textual string which describes the function status. The string is NULL terminated like any other C language string. The number of bytes returned is the smaller of SizeofString and the number of bytes contained in the actual description.
Values	Textual string which describes the function status

Example

C

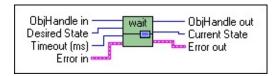
Check the status returned from the ncOpenDnetIntf function, and if not success, print a descriptive string.

ncWaitForState (Wait For State) Purpose

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

Format

LabVIEW



C

NCTYPE_STATUS

ncWaitForState(
NCTYPE_OBJH ObjHandle,
NCTYPE_STATE DesiredState,
NCTYPE_DURATION Timeout,
NCTYPE_STATE_P CurrentState)

Input

ObjHandle Object handle of an open Explicit Messaging Object or

an I/O Object

DesiredState States to wait for

Timeout Number of milliseconds to wait for one of the desired

states

Output

CurrentState Current state of object

Function Description

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

ncWaitforState is commonly used to wait for the Established state of an Explicit Messaging Object, or else to wait for an explicit message response resulting from a call to ncWriteDnetExplMsg, then read that response using ncReadDnetExplMsg.

While waiting for the desired states, ncWaitForState suspends the current execution. For C, this could suspend your front panel user interface. For LabVIEW, you can still access your front panel and the functions that are not directly connected to ncWaitForState can still execute. If you want to allow other code in your application to execute while waiting for NI-DNET states, refer to the ncCreateNotification (C only) function.

The functions ncWaitForState and ncCreateNotification use the same underlying implementation. Therefore, for each object handle, only one of these functions can be pending at a time. For example, you cannot invoke ncWaitForState twice from different threads for the same object. For different object handles, these functions can overlap in execution.

The status returned from ncWaitForState indicates any error detected by NI-DNET. You should always check this return status prior to checking the CurrentState value returned from ncWaitForState.

Parameter Descriptions

ObjHandle

•	ObjHandle must contain an object handle returned from ncOpenDnetExplMsg or ncOpenDnetIO.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

DesiredState

Description States to wait for. Each state is represented by a single bit so that you can wait for multiple states simultaneously. For example, if NI-DNET provides states with values of hex 1 and hex 4. DesiredState of hex 5 waits for either state to occur.

ReadAvail for the I/O Object

For the I/O Object, the ReadAvail state is set when a new input message is received from the network. The ReadAvail state clears when you call ncReadDnetIO. For example, for a change-of-state (COS) I/O connection, the ReadAvail state sets when a COS input message is received.

Although you can use ncWaitForState with an I/O Object, it is often preferable to use a notification (ncCreateNotification, C only). Use of a notification callback for the ReadAvail state allows your application to handle multiple I/O connections independently.

ReadAvail for the Explicit Messaging Object

For the Explicit Messaging Object, the ReadAvail state sets when an explicit message response is received from the network. The ReadAvail state clears when you call ncReadDnetExplMsg. An explicit message response is received only after you send an explicit message request using ncWriteDnetExplMsg. The following sequence of calls is typical: ncWriteDnetExplMsg, ncWaitForState,

ncReadDnetExplMsg. This sequence is used internally by ncGetDnetAttribute and ncSetDnetAttribute.

The ReadAvail state is not needed when using the explicit messaging functions ncGetDnetAttribute and ncSetDnetAttribute because both of these functions wait for the explicit message response internally.

Established for the Explicit Messaging Object

For the Explicit Messaging Object, the Established state is clear (not established) before you start communication using ncOperateDnetIntf. After you start communication, the Established state remains clear until the explicit message connection has been successfully established with the remote DeviceNet device. After the explicit message connection has been established, the Established state sets and remains set for as long as the explicit message connection is open.

Until the Established state sets for the Explicit Messaging Object, all calls to ncGetDnetAttribute, ncSetDnetAttribute, or ncWriteDnetExplMsg return the error CanErrNotStarted. Before you call any of these functions in your application, you must first wait for the Established state to set.

After the Established state is set, unless communication problems occur with the device (CanErrFunctionTimeout), it remains set until you stop communication using ncOperateDnetIntf.

While waiting for one of the above states, if an error occurs (such as a communication error or an initialization error), the wait returns immediately with the appropriate error code. For example, if you call ncWaitforState with DesiredState of ReadAvail, the wait function will return when data is available for a read, or when a DeviceNet communication error (such as connection timeout) is detected.

Values

A combination of one or more of the following bit values.

1 hex (ReadAvail, constant NC_ST_READ_AVAIL)

8 hex (Established, constant NC_ST_ESTABLISHED)
In LabVIEW and the LabWindows/CVI function panel, to facilitate combining multiple states, you can select a valid combination from an enumerated list of all valid combinations. This list contains the names of each state in the combination, such as ReadAvail or Established.

Timeout

	Number of milliseconds to wait for one of the desired states. If the timeout expires before one of the desired states occurs, ncWaitForState returns a status of BFF62001 hex (CanErrFunctionTimeout). The special timeout value of FFFFFFFF hex is used to wait indefinitely.
	indefinitely.
Values	1 to 200000 or FFFFFFFF hex (infinite duration, constant NC_DURATION_INFINITE)

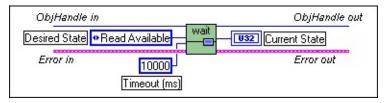
CurrentState

Description	Current state of the object. If one of the desired states occurs, it provides the current value of the ReadAvail and Established states. If the Timeout expires before one of the desired states occurs, it has the value 0.
Values	0 (desired states did not occur) or
	A combination of one or more of the following bit values.
	1 hex (ReadAvail, constant NC_ST_READ_AVAIL)
	8 hex (Established, constant NC_ST_ESTABLISHED)

Examples

LabVIEW

Wait up to 10 seconds for the ReadAvail state of an Explicit Messaging Object.



C

Wait up to 10 seconds for the ReadAvail state of the Explicit Messaging Object referenced by objh.

```
NCTYPE_STATUS status;

NCTYPE_OBJH objh;

NCTYPE_STATE currstate;

status = ncWaitForState(objh, NC_ST_READ_AVAIL, 10000, &currstate);
```

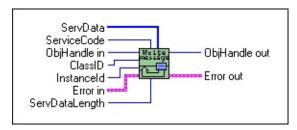
ncWriteDnetExplMsg (Write DeviceNet Explicit Message)

Purpose

Write an explicit message request using an Explicit Messaging Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncWriteDnetExplMsg(

NCTYPE_OBJH ObjHandle,

NCTYPE_UINT8 ServiceCode,

NCTYPE_UINT16 ClassId, NCTYPE_UINT16 InstanceId,

NCTYPE_UINT16 ServDataLength,

NCTYPE_ANY_P ServData);

Input

ObjHandle Object handle of an open Explicit Messaging Object

ServiceCode Identifies the service being requested

ClassId Identifies the class to which service is directed

InstanceId Identifies the instance to which service is directed

ServDataLength Number of service data bytes for request

ServData Service data for request

Output

None

Function Description

ncWriteDnetExplMsg writes an explicit message request using an Explicit Messaging Object.

The two most commonly used DeviceNet explicit messages are the Get Attribute Single service and the Set Attribute Single service. The easiest way to execute the Get Attribute Single service on a remote device is to use the NI-DNET ncGetDnetAttribute function. The easiest way to execute the Set Attribute Single service on a remote device is to use the NI-DNET ncSetDnetAttribute function.

To execute services other than Get Attribute Single and Set Attribute Single, use the following sequence of function calls: ncWriteDnetExplMsg, ncWaitForState, ncReadDnetExplMsg. The ncWriteDnetExplMsg function sends an explicit message request to a remote DeviceNet device. The ncWaitForState function waits for the explicit message response, and the ncReadDnetExplMsg function reads that response.

Some DeviceNet services that use ncWriteDnetExplMsg are Reset, Save, Restore, Get Attributes All, and Set Attributes All. Although the *DeviceNet Specification* defines the overall format of these services, in most cases their meaning and service data are object specific or vendor specific. Unless your device requires such services and documents them in detail, you probably do not need them for your application. For more information, refer to the *NI-DNET User Manual*.

Parameter Descriptions

ObjHandle

•	ObjHandle must contain an object handle returned from ncOpenDnetExplMsg.
	In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

ServiceCode

·	Identifies the service being requested. You can find service code values for the commonly used DeviceNet services in the DeviceNet Specification (Volume 1, Appendix G, DeviceNet Explicit Messaging Services). The device's vendor documents vendor-specific service codes.
Values	00 to FF hex

ClassId

Description	Identifies the class to which service is directed. You can find descriptions and identifiers for each standard DeviceNet class in the <i>DeviceNet Specification</i> (Volume 2, Chapter 6, <i>The DeviceNet Object Library</i>). The device's vendor documents vendor-specific classes. Although the <i>DeviceNet Specification</i> allows 16-bit class IDs, most class IDs are 8-bit. NI-DNET automatically uses the class ID size (16-bit or 8-bit) that is appropriate for your device
	8-bit) that is appropriate for your device.
Values	00 to FFFF hex

InstanceId

Description	Identifies the instance to which service is directed. Instance
	ID 0 is used to direct the service toward the class itself.
	Other instance IDs typically are numbered starting at 1. For
	example, the primary Identity Object in a device uses
	instance ID 1. Although the <i>DeviceNet Specification</i> allows
	16-bit instance IDs, most instance IDs are 8-bit. NI-DNET
	automatically uses the instance ID size (16-bit or 8-bit) that

	is appropriate for your device.
Values	00 to FFFF hex

${\bf ServDataLength}$

<u> </u>	Number of service data bytes for the request. This length also specifies the number of bytes provided in ServData.
Values	0 to 240

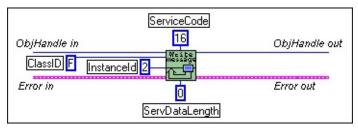
ServData

Description	Service data bytes for the request. The format of this data is specific to the service code being used. For commonly used services which are not object specific, the format of this data is defined in the <i>DeviceNet Specification</i> (Volume 1, Appendix G, <i>DeviceNet Explicit Messaging Services</i>). For object-specific service codes, the format of this data is defined in the object specification. For vendor-specific service codes, the format of this data is defined by the device vendor. The ServDataLength parameter specifies the number of service data bytes sent in the request (and provided in this buffer).
Values	Service data bytes for the request

Examples

LabVIEW

Save the parameters of Parameter Object instance 2 to non-volatile memory. The service code for Save is 16 hex. The Parameter Object is class ID 0F hex. The Parameter Object does not define any service data bytes for Save.



C

Reset a DeviceNet device to its power on state using the Explicit Messaging Object referenced by objh. The service code for Reset is 05 hex. The Identity Object (class ID 1, instance ID 1) is used to reset DeviceNet devices. The Identity Object defines a single byte of service data, where 0 is used to simulate a power cycle and 1 is used to reset the device to its out-of-box state.

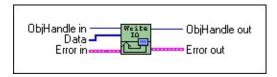
```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
NCTYPE_UINT8 type_of_reset;
type_of_reset = 0;
status = ncWriteDnetExplMsg(objh, 0x05, 0x01, 0x01, 1, &type_of_reset);
```

ncWriteDnetIO (Write DeviceNet I/O) Purpose

Write output data to an I/O Object.

Format

LabVIEW



C

NCTYPE_STATUS

ncWriteDnetIO(NCTYPE_OBJH ObjHandle, NCTYPE_UINT32 SizeofData, NCTYPE_ANY_P Data);

Input

ObjHandle Object handle of an open I/O Object

SizeofData Size of Data buffer in bytes (C only)

Data Output data

Output

None

Function Description

ncWriteDnetIO writes output data to an NI-DNET I/O Object.

Since each I/O Object continuously produces output data onto the DeviceNet network at a specified rate, calling ncWriteDnetIO multiple times for each output message is redundant and can often waste valuable processor time. To synchronize calls to ncWriteDnetIO with each output message, you can wait for input data (see ncReadDnetIO), or if no input data exists for the device, you can use an idle wait (such as wait for 10 ms).

The output data bytes passed to ncWriteDnetIO are normally sent to the output assembly of a remote DeviceNet slave device. The format of this output assembly is normally documented either by the device vendor or within the *DeviceNet Specification* itself.

The bytes of a device's output assembly often consist of multiple data members rather than a single value. For C, you can often place each data member into the output bytes by using typecasting. For LabVIEW, you can often place each data member into the output bytes using the ncConvertForDnetWrite function. For more information on output assemblies and how to place individual data members into the output bytes, refer to the *NI-DNET User Manual*.

Parameter Descriptions

ObjHandle

	ObjHandle must contain an object handle returned from ncOpenDnetIO. In LabVIEW, ObjHandle passes through the VI as an output so that it can be used for subsequent function calls for the object.
Values	The encoding of ObjHandle is internal to NI-DNET.

SizeofData

Description	For C, SizeofData is the size of the buffer referenced by Data. It is used to verify that the Data buffer is large enough to hold the output bytes. This size is normally obtained using the C language sizeof function and has no direct relation to the number of bytes produced on the network.
	For LabVIEW, since Data is obtained directly as an input, this size is not needed.
	The actual number of bytes produced on the I/O connection is determined by the OutputLength parameter of ncOpenDnetIO and not this size.
Values	sizeof (buffer referenced by Data)

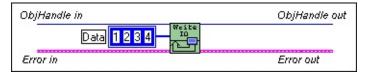
Data

•	Output data. The format of these output bytes is specific to your DeviceNet device.
Values	Output data bytes

Examples

LabVIEW

Write 4 output bytes to an I/O Object.



C

Write 10 output bytes to the I/O Object referenced by objh.

```
NCTYPE_STATUS status;
NCTYPE_OBJH objh;
NCTYPE_UINT8 output[10];
status = ncWriteDnetIO(objh, 10, output);
```

NI-DNET Objects

The Explicit Messaging Object, Interface Object, and I/O Object topics describe each NI-DNET object, list the functions which can be used with the object, and describe each of the object's driver attributes. The description of each object is structured as follows:

Description

Gives an overview of the major features and uses of the object.

Functions

Lists each NI-DNET function which can be used with the object. For information on how each NI-DNET function is used with the object, refer to <u>NI-DNET Functions</u>.

Driver Attributes

Lists and describes the NI-DNET driver attributes for each object. The driver attributes are listed in alphabetical order.

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

- Get You can get the attribute at any time using ncGetDriverAttr, but never set it.
- Set You can get the attribute at any time using ncGetDriverAttr. You can set the attribute using ncSetDriverAttr, but only prior to starting communication using ncOperateDnetIntf.

Explicit Messaging Object

Description

The Explicit Messaging Object represents an explicit messaging connection to a remote DeviceNet device (physical device attached to your interface by a DeviceNet cable). Since only one explicit messaging connection is created for a given device, the Explicit Messaging Object is also used for features that apply to the device as a whole.

Use the Explicit Messaging Object to do the following:

- Execute the DeviceNet Get Attribute Single service on the remote device (ncGetDnetAttribute).
- Execute the DeviceNet Set Attribute Single service on the remote device (ncSetDnetAttribute).
- Send any other explicit message requests to the remote device and receive the associated explicit message response (ncWriteDnetExplMsg, ncReadDnetExplMsg).
- Configure NI-DNET settings that apply to the entire remote device.

Functions

Function Name	Function Description
ncCloseObject	Close an NI-DNET object
ncConvertForDnetWrite	Convert an appropriate LabVIEW data type for writing data bytes on the DeviceNet network
ncConvertFromDnetRead	Convert data read from the DeviceNet network into an appropriate LabVIEW data type
ncCreateNotification	Create a notification callback for an object (C only)
ncGetDnetAttribute	Get an attribute value from a DeviceNet device
ncGetDriverAttr	Get the value of an attribute in the NI-DNET driver
ncOpenDnetExplMsg	Configure and open an NI-DNET Explicit Messaging Object
ncReadDnetExplMsg	Read an explicit message response
ncSetDnetAttribute	Set an attribute value for a DeviceNet device
ncSetDriverAttr	Set the value of an attribute in the NI-DNET driver
ncStatusToString	Convert status returned from an NI-DNET function into a descriptive string (C only)
ncWaitForState	Wait for one or more states to occur in an object
ncWriteDnetExplMsg	Write an explicit message request

Driver Attributes

Current State

Attribute ID	NC_ATTR_STATE
Hex Encoding	8000009
Data Type	NCTYPE_STATE
Permissions	
Description	Current state of the NI-DNET object. This driver attribute provides the current ReadAvail and Established states as described in the ncWaitForState function.
	Use ncGetDriverAttr when you need to determine the current state of an object but you do not need to wait for a specific state.

Device Type

Attribute ID	NC_ATTR_DEVICE_TYPE
Hex Encoding	80000084
Data Type	NCTYPE_UINT16
Permissions	Set
Description	Device Type of the device as reported in the Device Type attribute of device's Identity Object. This attribute verifies that the device is the same one expected by your application. If the Device Type does not match, NI-DNET returns the error DnetErrDevInitDevType. The Device Type indicates conformance to a specific device profile, such as Photoelectric Sensor or Position Controller.
	If you do not call ncSetDriverAttr to set the Device Type, a default value of zero is used. When Device Type is zero, NI-DNET does not verify the device's Device Type.

Keep Explicit Messaging

Attribute ID	NC_ATTR_KEEP_EXPL_MSG
Hex Encoding	80000099
Data Type	NCTYPE_BOOL
Permissions	Set
Description	To properly close I/O connections in the remote device when ncCloseObject is called, NI-DNET must ensure that an explicit messaging connection to the device remains open. When this attribute is set to NC_TRUE (the default), NI-DNET sends a nonoperational request (a "ping") to the device every few seconds, to ensure that the explicit messaging connection does not timeout. When this attribute is NC_FALSE, NI-DNET does not ping the explicit messaging connection. If you are certain that your application sends a request on a periodic basis, you can set this attribute to NC_FALSE. This attribute must be set prior to starting communication.

Mac Id

Attribute ID	NC_ATTR_MAC_ID
Hex Encoding	80000080
Data Type	NCTYPE_UINT8
Permissions	Get
Description	This driver attribute allows you to get the DeviceMacId originally passed into ncOpenDnetExplMsg.

Product Code

Attribute ID	NC_ATTR_PRODUCT_CODE
Hex	80000083
Encoding	
Data Type	NCTYPE_UINT16
Permissions	Set
Description	Product Code of the device as reported in the Product

Code attribute of device's Identity Object. This attribute verifies that the device is the same one expected by your application. If the Product Code does not match, NI-DNET returns the error DnetErrDevInitProdCode.

The Product Code is a vendor-specific value which identifies a particular product within a device type.

If you do not call ncSetDriverAttr to set the Product Code, a default value of zero is used. When Product Code is zero, NI-DNET does not verify the device's Product Code.

Vendor Id

Attribute ID	NC_ATTR_VENDOR_ID
Hex	80000082
Encoding	
Data Type	NCTYPE_UINT16
Permissions	Set
Description	Vendor ID of the device as reported in the Vendor ID attribute of device's Identity Object. This attribute verifies that the device is the same one expected by your application. If the Vendor ID does not match, NI-DNET returns the error DnetErrDevInitVendor. The Vendor ID is a number assigned to the device vendor by the Open Device Vendor's Association (ODVA). If you do not call ncSetDriverAttr to set the Vendor ID, a default value of zero is used. When Vendor ID is zero, NI-DNET does not verify the device's Vendor ID.

Interface Object

Description

The Interface Object represents a DeviceNet interface. Since this interface acts as a device on the DeviceNet network much like any other device, it is configured with its own MAC ID and baud rate.

Use the Interface Object to do the following:

- Configure NI-DNET settings that apply to the entire interface.
- Start and stop communication for all NI-DNET objects associated with the interface.

The Interface Object must be the first NI-DNET object opened by your application, and thus the ncOpenDnetIntf function must be the first NI-DNET function called by your application.

Functions

Function Name	Function Description
EasyIOClose	Close multiple NI-DNET objects (LabVIEW only)
EasyIOConfig	Configure and open multiple NI-DNET objects (LabVIEW only)
ncCloseObject	Close an NI-DNET object
ncGetDriverAttr	Get the value of an attribute in the NI-DNET driver
ncOpenDnetIntf	Configure and open an NI-DNET Interface Object
ncOperateDnetIntf	Perform an operation on an NI-DNET Interface Object
ncSetDriverAttr	Set the value of an attribute in the NI-DNET driver
ncStatusToString	Convert status returned from an NI-DNET function into a descriptive string (C only)

Driver Attributes

Baud Rate

Attribute ID	NC_ATTR_BAUD_RATE
Hex Encoding	8000007
Data Type	NCTYPE_BAUD_RATE
Permissions	Get
Description	This driver attribute allows you to get the BaudRate originally passed into ncOpenDnetIntf.

Interface Protocol Version

Attribute ID	NC_ATTR_PROTOCOL_VERSION
Hex	8000002
Encoding	
Data Type	NCTYPE_VERSION
Permissions	Get
Description	This driver attribute reports the version of the <i>DeviceNet Specification</i> to which the NI-DNET software conforms. This version is at least 02000000 hex (version 2.0).

Interface Software Version

Attribute ID	NC_ATTR_SOFTWARE_VERSION
Hex Encoding	8000003
Data Type	NCTYPE_VERSION
Permissions	Get
Description	This driver attribute reports the version of the NI-DNET software. This version is at least 01000000 hex (version 1.0).

Mac Id

Attribute ID	NC_ATTR_MAC_ID
Hex	80000080
Encoding	

Data Type	NCTYPE_UINT8
Permissions	Get
-	This driver attribute allows you to get the IntfMacId originally passed into ncOpenDnetIntf.

Poll Mode

Attribute ID	NC_ATTR_POLL_MODE
Hex Encoding	8000009B
Data Type	NCTYPE_POLL_MODE
Permissions Get	
Description	This driver attribute allows you to get the PollMode originally passed into ncOpenDnetIntf.

I/O Object

Description

The I/O Object represents an I/O connection to a remote DeviceNet device (physical device attached to your interface by a DeviceNet cable). The I/O Object usually represents I/O communication as a master with a remote slave device. If your computer is being used as the primary controller of your DeviceNet devices, you should configure I/O communication as a master.

You can also configure the I/O Object for I/O communication as a slave with a remote master. If your computer is being used as a peripheral device for another primary controller, you can configure I/O communication as a slave. To configure I/O communication as a slave, set the I/O Object's DeviceMacId to the same MAC ID as the Interface Object (IntfMacId parameter of ncOpenDnetIntf).

The I/O Object supports as many master/slave I/O connections as currently allowed by the *DeviceNet Specification* (version 2.0). This means that you can use polled, strobed, and COS/cyclic I/O connections simultaneously for a given device. As specified by the *DeviceNet Specification*, only one master/slave I/O connection of a given type can be used for each device (MAC ID). For example, you cannot open two polled I/O connections for the same device.

Use the I/O Object to do the following:

- Read data from the most recent message received on the I/O connection (ncReadDnetIO).
- Write data for the next message produced on the I/O connection (ncWriteDnetIO).

Functions

Function Name	Function Description
EasyIOClose	Close multiple NI-DNET objects (LabVIEW only)
EasyIOConfig	Configure and open multiple NI-DNET objects (LabVIEW only)
ncCloseObject	Close an NI-DNET object
ncConvertForDnetWrite	Convert an appropriate LabVIEW Data Type for writing data bytes on the DeviceNet network
ncConvertFromDnetRead	Convert data read from the DeviceNet network into an appropriate LabVIEW Data Type
ncCreateNotification	Create a notification callback for an object (C only)
ncGetDriverAttr	Get the value of an attribute in the NI-DNET driver
ncOpenDnetIO	Configure and open an NI-DNET I/O Object
ncReadDnetIO	Read input data from an I/O Object
ncSetDriverAttr	Set the value of an attribute in the NI-DNET driver
ncStatusToString	Convert status returned from an NI-DNET function into a descriptive string (C only)
ncWaitForState	Wait for one or more states to occur in an object
ncWriteDnetIO	Write output data to an I/O Object

Driver Attributes

Ack Suppress

Attribute ID	NC_ATTR_ACK_SUPPRESS
Hex Encoding	8000009A
Data Type	NCTYPE_BOOL
Permissions	Set
Description	This driver attribute applies only to change-of-state (COS) or cyclic I/O connections (ConnectionType of COS or Cyclic). It determines whether acknowledgments are used (false) or suppressed (true). Acknowledgments are used with COS or cyclic I/O connections to verify that produced data is received successfully.
	When InputLength is nonzero, the acknowledgment is produced by NI-DNET. When OutputLength is nonzero, the acknowledgment is consumed by NI-DNET.
	If you do not call ncSetDriverAttr to Set Ack Suppress, a default value of false is used.
	When successful device operation can be verified by other means, COS or cyclic acknowledgment can often be suppressed. For example, if you open a polled I/O connection in addition to the COS or cyclic I/O connection, you can Set Ack Suppress to true.
	If the ConnectionType of this I/O object is Poll or Strobe, the Ack Suppress attribute is ignored.

Current State

Attribute ID	NC_ATTR_STATE
Hex Encoding	8000009
Data Type	NCTYPE_STATE

Permissions	Get
	Current state of the NI-DNET object. This driver attribute provides the current ReadAvail and Established states as described in ncWaitForState.

Device Type

Attribute ID	NC_ATTR_DEVICE_TYPE
Hex	80000084
Encoding	
Data Type	NCTYPE_UINT16
Permissions	Set
Description	Device Type of the device as reported in the Device Type attribute of device's Identity Object. This attribute verifies that the device is the same one expected by your application. If the Device Type does not match, NI-DNET returns the error DnetErrDevInitDevType. The Device Type indicates conformance to a specific device profile, such as Photoelectric Sensor or Position Controller.
	If you do not call ncSetDriverAttr to Set the Device Type, a default value of zero is used. When Device Type is zero, NI-DNET does not verify the device's Device Type.

Exp Packet Rate

Attribute ID	NC_ATTR_EXP_PACKET_RATE
Hex Encoding	80000095
Data Type	NCTYPE_DURATION
Permissions	Get
Description	This driver attribute allows you to get the ExpPacketRate originally passed into ncOpenDnetIO.

Inhibit Timer

4	Attribute ID	NC_ATTR_EXP_INHIBIT_TIMER
Γ		

Hex Encoding	80000097
Data Type	NCTYPE_DURATION
Permissions	Set
Description	This driver attribute applies only to COS I/O connections (ncOpenDnetIO with ConnectionType of COS). This driver attribute configures the minimum delay time between subsequent data productions. This attribute can limit the amount of network traffic used for COS messages from devices with frequently changing I/O. The default value for Inhibit Timer is zero, as specified in the <i>DeviceNet Specification</i> . Since this default is appropriate for most applications, the Inhibit Timer attribute is not included in the configuration attributes provided with ncOpenDnetIO. If you want to change the default Inhibit Timer, call ncSetDriverAttr prior to starting communication. If ConnectionType is Poll, Strobe, or Cyclic, the Inhibit Timer attribute is ignored. For these I/O connection types, the frequency of data production is controlled entirely by the ExpPacketRate attribute.

Input Length

Attribute ID	NC_ATTR_IN_LEN
Hex Encoding	8000091
Data Type	NCTYPE_UINT32
Permissions	Get
Description	This driver attribute allows you to get the InputLength originally passed into ncOpenDnetIO.

Keep Explicit Messaging

Attribute ID	NC_ATTR_KEEP_EXPL_MSG
Hex Encoding	80000099

Data Type	NCTYPE_BOOL
Permissions	Set
Description	To properly close I/O connections in the remote device when ncCloseObject is called, NI-DNET must ensure that an explicit messaging connection to the device remains open. When this attribute is Set to NC_TRUE (the default), NI-DNET sends a nonoperational request (a "ping") to the device every few seconds, to ensure that the explicit messaging connection does not timeout. When this attribute is NC_FALSE, NI-DNET does not ping the explicit messaging connection. If you are certain that your device can internally close its own I/O connections (deferred delete), you can Set this attribute to NC_FALSE. This attribute must be Set prior to starting communication.

Mac Id

Attribute ID	NC_ATTR_MAC_ID
Hex	80000080
Encoding	
Data Type	NCTYPE_UINT8
Permissions	Get
Description	This driver attribute allows you to get the DeviceMacId originally passed into ncOpenDnetIO.

Output Length

Attribute ID	NC_ATTR_OUT_LEN
Hex	80000092
Encoding	
Data Type	NCTYPE_UINT32
Permissions	Get
Description	This driver attribute allows you to get the OutputLength originally passed into ncOpenDnetIO.

Product Code

Attribute ID	NC_ATTR_PRODUCT_CODE
Hex	80000083

Encoding	
Data Type	NCTYPE_UINT16
Permissions	Set
Description	Product Code of the device as reported in the Product Code attribute of device's Identity Object. This attribute is used to verify that the device is the same one expected by your application. If the Product Code does not match, NI-DNET returns the error DnetErrDevInitProdCode. The Product Code is a vendor-specific value which identifies a particular product within a device type. If you do not call ncSetDriverAttr to Set the Product Code, a default value of zero is used. When Product Code is zero, NI-DNET does not verify the device's Product Code.

Vendor Id

Attribute ID	NC_ATTR_VENDOR_ID
Hex Encoding	80000082
Data Type	NCTYPE_UINT16
Permissions	
Description	Vendor ID of the device as reported in the Vendor ID attribute of device's Identity Object. This attribute verifies that the device is the same one expected by your application. If the Vendor ID does not match, NI-DNET returns the error DnetErrDevInitVendor. The Vendor ID is a number assigned to the device vendor by the Open Device Vendor's Association (ODVA). If you do not call ncSetDriverAttr to Set the Vendor ID, a default value of zero is used. When Vendor ID is zero, NI-DNET does not verify the device's Vendor ID.

Glossary

Prefixes Numbers/Symbols A B C D E H I K L M

N O P R S V

Prefixes

Symbol	Prefix	Value
р	pico	10 -12
n	nano	10 -9
μ	micro	10 -6
m	milli	10 -3
k	kilo	10 ³
М	mega	10 ⁶
G	giga	10 ⁹
Т	tera	10 12

Numbers/Symbols

nV	nanovolts	10 ⁻⁹ volts
μV	microvolts	10 ⁻⁶ volts
μΩ	microohms	10 ⁻⁶ ohms
mΩ	milliohms	10 ⁻³ ohms
МΩ	megaohms	10 ⁶ ohms
рА	picoamps	10 ⁻¹² amperes
nA	nanoamps	10 ⁻⁹ amperes
μΑ	microamps	10 ⁻⁶ amperes
mA	milliamps	10 ⁻³ amperes

Α

ANSI American National Standards Institute.

Application A collection of functions used by a user application to Programming access hardware. Within NI-DNET, you use API functions

Interface to make calls into the NI-DNET driver.

(API)

ASCII American Standard Code for Information Interchange.

attribute The externally visible qualities of an object; for example,

an instance square of class geometric shapes could have the attributes *length of sides* and *color*, with the values 4

in. and blue. Also known as property.

В

b Bits.

bit Master/slave I/O connection in which the master broadcasts a strobed single strobe command to all strobed slaves, then receives a I/O strobe response from each strobed salve.

C

CAN Controller Area Network.

change-of- Master/slave I/O connection which is similar to cyclic I/O, state I/O but data can be sent when a change in the data is detected.

class A classification of things with similar qualities.

connection An association between two or more devices on a network that describes when and how data is transferred.

controller A device that receives data from sensors and sends data to

actuators to hold one or more external, real-world variables

at a certain level or condition. A thermostat is a simple

example of a controller.

COS I/O See change-of-state I/O.

cyclic I/O Master/slave I/O connection in which the slave (or master)

sends data at a fixed interval.

D

device A physical assembly, linked to a communication line (cable),

capable of communicating across the network according to a

protocol specification.

device Multi-drop digital communication network for sensors,

network actuators, and controllers.

DeviceNet A physical DeviceNet port on an AT-CAN, PCI-CAN,

interface PCMCIA-CAN, or PXI-8461 interface.

Ε

expected packet rate explicit messaging connection

The rate (in milliseconds) at which a DeviceNet connection is expected to transfer its data.

General-purpose connection used for executing services on a particular object in a DeviceNet device.

Н

hex Hexadecimal.

I/O Connection used for exchange of physical input/output connection (sensor/activator) data, as well as other control-oriented data.

individual A polled I/O communication scheme in which each polled polling slave communicates at its own individual rate.

instance A specific instance of a given class. For example, a blue square of 4 inches per side would be one instance of the class Geometric Shapes.

K

KB Kilobytes of memory.

L

LabVIEW Laboratory Virtual Instrument Engineering Workbench.

local Within NI-DNET, anything that exists on the same host (personal computer) as the NI-DNET driver.

M

MAC ID Media access control layer identifier. In DeviceNet, a

device's MAC ID represents its address on the DeviceNet

network.

master/slave DeviceNet communication scheme in which a master

device allocates connections to one or more slave

devices, and those slave devices can only communicate

with the master and not one another.

member An individual data value within an array of DeviceNet data

bytes.

method An action performed on an instance to affect its behavior;

the externally visible code of an object. Within NI-DNET,

you use NI-DNET functions to execute methods for objects. Also known as service, operation, and action.

multi-drop A physical connection in which multiple devices

communicate with one another along a single cable.

Ν

NI-DNET Device driver and/or firmware that implement all the driver specifics of a National Instruments DeviceNet interface.

notification Within NI-DNET, an operating system mechanism that the NI-DNET driver uses to communicate events to your application. You can think of a notification of as an API function, but in the opposite direction.

0

object See <u>instance</u>.

ODVA Open DeviceNet Vendor's Association.

Ρ

olled Master/slave I/O connection in which the master sends a poll I/O command to a slave, then receives a poll response from that slave.

protocol A formal set of conventions or rules for the exchange of information among devices of a given network.

R

remote Within NI-DNET, anything that exists in another device of the device network (not on the same host as the NI-DNET driver).

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

s Seconds.

scanned A polled I/O communication scheme in which all poll polling commands are sent out at the same rate, in quick succession.

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*.

strobed See <u>bit strobed I/O</u>. I/O

٧

VI Virtual Instrument.

Important Information

Warranty

Copyright

Trademarks

<u>Patents</u>

Warning Regarding Use of NI Products

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

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

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

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about <u>National Instruments trademarks</u>.

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, and Stateflow® are registered trademarks, and TargetBox™, xPC TargetBox™, and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

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

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

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your CD, or <u>ni.com/patents</u>.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

- (1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.
- (2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS. SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE. INJURY. OR DEATH. THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES. INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR

APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- <u>Support</u>—Online technical support resources at ni.com/support include the following:
 - Self-Help Resources—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable <u>KnowledgeBase</u>, <u>product</u> <u>manuals</u>, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - Free Technical Support—All registered users receive free Basic Service, which includes access to hundreds of Applications Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Applications Engineers make sure every question receives an answer.

For information about other <u>technical support options</u> in your area, visit ni.com/services or <u>contact</u> your local office at ni.com/contact.

- <u>Training and Certification</u>—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- <u>System Integration</u>—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your <u>local office</u> or NI corporate headquarters. You also can visit the <u>Worldwide Offices</u> section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Branch Offices

Office	Telephone Number
Australia	1800 300 800
Austria	43 0 662 45 79 90 0
Belgium	32 0 2 757 00 20
Brazil	55 11 3262 3599
Canada	800 433 3488
China	86 21 6555 7838
Czech Republic	420 224 235 774
Denmark	45 45 76 26 00
Finland	385 0 9 725 725 11
France	33 0 1 48 14 24 24
Germany	49 0 89 741 31 30
India	91 80 41190000
Israel	972 0 3 6393737
Italy	39 02 413091
Japan	81 3 5472 2970
Korea	82 02 3451 3400
Lebanon	961 0 1 33 28 28
Malaysia	1800 887710
Mexico	01 800 010 0793
Netherlands	31 0 348 433 466
New Zealand	0800 553 322
Norway	47 0 66 90 76 60
Poland	48 22 3390150
Portugal	351 210 311 210
Russia	7 095 783 68 51
Singapore	1800 226 5886
Slovenia	386 3 425 4200

South Africa	27 0 11 805 8197
Spain	34 91 640 0085
Sweden	46 0 8 587 895 00
Switzerland	41 56 200 51 51
Taiwan	886 02 2377 2222
Thailand	662 278 6777
United Kingdom	44 0 1635 523545
United States (Corporate)	512 683 0100