Target

DeviceNet API
for
esd-CAN-Boards

Software Manual

esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
http://www.esd.eu
Phone: +49 (0) 511 3 72 98-0 • Fax: +49 (0) 511 3 72 98-68

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. In particular descriptions and technical data specified in this document may not be constituted to be guaranteed product features in any legal sense.

esd reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd.eu
Internet: www.esd.eu

Manual File:	I:\Texte\Doku\MANUALS\PROGRAM\CAN\DeviceN\TrgetUni\DeviceNet_API-Manual_en_17.wpd
Date of Print:	2016-01-22

Described Software	**Revision/Date**
Local DeviceNet API-Library	Windows 32/64-Bit: V2.2.0 / 2011-08-23
	VxWorks: V2.2.2 / 2004-05-28

Implementation at the Following Boards	**Order no.**
DN-ISA/331-11x DeviceNet DN-ISA/331-22x DeviceNet	C.2016.02 C.2016.04
DN-PCI/331-11x DeviceNet DN-PCI/331-22x DeviceNet	C.2017.06 C.2017.07
DN-PC-104/331 1x DeviceNet	C.2014.02
DN-VME-CAN/4 4x DeviceNet	V.1408.08

Changes in the chapters

The changes in the user's manual listed below affect changes in the software, as well as changes in the description of the facts only.

Version	Chapter	Alterations versus previous revisions	Alternations in software	Alternations in documentation
1.6	-	Description of function *dnetDataEvMask ( )* added.	-	x
1.7	-	Classification of notes inserted, notes revised	-	x
	6.	New chapter: “DeviceNet Errata DN-PCI/331" inserted	-	x

Technical details are subject to change without notice.

Classification of Notice Statements and Information

NOTICE
Notice statements are used to notify people on hazards that could result in things other than personal injury, like property damage or software error.

NOTICE
This NOTICE statement contains the general mandatory sign and gives information that must be heeded and complied with for a safe and error-free use.

INFORMATION

INFORMATION
Notes to point out something important or useful.

Contents

Page

1. Overview

The DeviceNet library is available for Windows and VxVorks. Further operating systems will be supported in the future. You will find the actual list of supported operating systems at the history page at the beginning of this manual.

The DeviceNet library supports as many DeviceNet channels as the underlying CAN driver provides to the DeviceNet library layer.

The following figure shows the structure of the software using the esd DN-ISA/331 as an example. The DeviceNet protocol handler program runs **locally** on the esd-CAN-board.

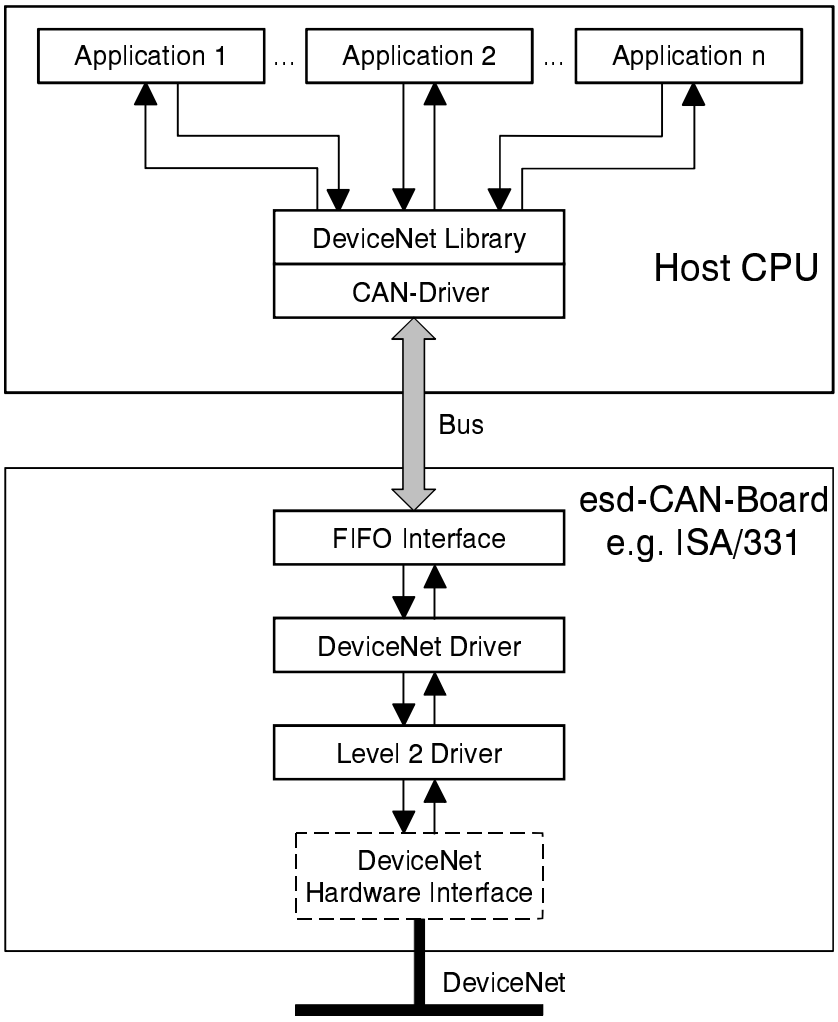

Fig. 1.1.1: Software structure

2. Function Description

2.1 Get Access to the Library Functions

dnetOpen( )

Name: **dnetOpen()** - Generates a handle to access the DeviceNet library

Synopsis:

```
STATUS    dnetOpen
          (
          int         net,
          HANDLE *handle
          )
```

Description: This function returns a handle that gives access to the other DeviceNet library functions.

Parameter:

net: logical net-no. (same number as CAN driver, defined in installation sequence)

handle*: handle initialized by ***dnetOpen ( )*

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

NOTICE

- To ensure the proper operation of the library every thread of execution (task) has to use its own handle. Or as a rule of thumb: only one thread per handle is allowed to enter the library at a time!
- Use a different handle at least for each independent running thread/task!
- If you are using a handle as event handle then you should use this handle only for calling ***dnetEvRead( )*** or ***dnetEvDisable( )***. Do **NOT** use it for any other calls!
- Only **ONE** event handle (***dnetEvEnable( )*** called with this handle) can exist for each network!
- The CAN driver allows at the moment a maximum of **8** DeviceNet handles per network. This is a compilation parameter and can easily be changed on demand.

dnetClose( )

Name: **dnetClose()** - Closing a handle of the DeviceNet library

Synopsis:

```
STATUS          dnetClose
(
HANDLE          handle
)
```

Description: This function closes the opened handle.

Parameter: *handle*: handle returned from ***dnetOpen ( )***

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.2 Hardware and DeviceNet Initialisation

dnetStart( )

Name: **dnetStart( )** - Initialize and start DeviceNet protocol

Synopsis:

```
STATUS   dnetStart
         (
         HANDLE handle,
         int    MACID,
         int    baudrate,
         int    rxLen,
         int    txLen,
         byte   option
         )
```

Description: This function initializes and starts the DeviceNet protocol on the network selected by *handle*. For the protocol the following parameters are necessary:

Parameter:

handle:	specifies the network to operate on	
MACID:	MACID of the local scanner and slave interface (0...63)	
baudrate:	bit rate	0 = 125 kbit/s
		1 = 250 kbit/s
		2 = 500 kbit/s
rxLen:	consumed data length of the slave interface (0...440)	
txLen:	produced data length of the slave interface (0...440)	
option:	bit 0: Event on receiving I/O data (local slave)	
	bit 1: Event on receiving changed I/O data (local slave)	

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetStop( )

Name: **dnetStop( )** - Remove remote slaves from scanner table

Synopsis:

```
STATUS    dnetStop
          (
          HANDLE handle
          )
```

Description: This function removes all remote slaves from the scanner table (its list of slaves to scan) and stops all CAN actions.

Parameter: *handle*: specifies the network to stop

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetInstRemSlave( )

Name: **dnetInstRemSlave( )** - Install a remote slave

Synopsis:

```
STATUS   dnetInstRemSlave
         (
HANDLE   handle,
int      MACID,
int      rxLen,
int      txLen,
int      EPR,
byte     allocation,
byte     option
         )
```

Description: This function installs a remote slave in the local scanner table. The firmware's local data image is initialized to zero. After that the scanner begins immediately to talk to this slave and maintains the requested connection to the slave.

Parameter:

handle: specifies the network to operate on

MACID: MACID of the remote slave interface (0...63)

rxLen: produced data length of the remote slave interface (0...440)

txLen: consumed data length of the remote slave interface (0...440)

EPR: The **e**xpected **p**ackage **r**ate of the slave (0...32767 ms). The scanner polls the slave with the half time of the EPR.
Limits: 0x3FFF (COS/Cyclic), 0x7FFF (Polling, BitStrobe)

allocation: Meaning is as described in the DeviceNet specification. An allocation value of 0 (zero) means to deinstall the remote slave!

Bit no.	7	6	5	4	3	2	1	0
Content	Reserved	Acknowledge Suppression	Cyclic	Change of State	Reserved	Bit Strobed	Polled	Explicit Message

Table 2.1.1: Allocation choice byte contents

The following choices for the allocation byte are possible:
- cyclic and/or change of state (COS) (0x20/0x10)
- bit strobed (0x04)
- polled (0x02)

Specify only one of these three. Additional selection of explicit message connection (0x01) is allowed

option: Mask 0x01: Event on receiving I/O data (remote slave)
Mask 0x02: Event on receiving changed I/O data (remote slave)
Mask 0x04: Generate event if receiving data changed in the valid bits area of the event mask (see ***dnetDataEvMask( )***)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetDataEvMask( )

Name: **dnetDataEvMask( )** -Modify data change event mask

Synopsis:

```
STATUS    dnetDataEvMask
          (
HANDLE    handle,
int       MACID,
int       offset,
int       len,
void      *dataEvMask
          )
```

Description: This function modifies the data change event mask for a remote slave. After a call of ***dnetInstRemSlave( )*** with *option* set to mask 0x04 (event on receiving changed data) a data event mask is established and initialised to all bytes 0xff. That means all bytes of the slave's incoming data are checked for changes.

You can set a new data event mask with bitwise granularity using this function. Any cleared bit of the mask means that the correspondent bit of the slave's incoming data is ignored when the firmware checks for changes.

Parameter:

handle:	specifies the network to operate on
MACID:	MACID of the remote slave interface
offset:	position of the data inside the change mask
len:	length of data to modify
**dataEvMask*:	points to buffer with new event mask

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetIdent( )

Name: **dnetIdent( )** -Read board information

Synopsis:

```
STATUS    dnetIdent
          (
          HANDLE      handle,
          DN_IDENT    *ident
          )
```

Description: This function gets information out of the Identity class of the DeviceNet board. Also it gets the firmware release number. Look at the typedef for DN_IDENT in the `dnet.h` file for further information.

Parameter:

handle: specifies the network to operate on

**ident*: Information about the Identity class of DeviceNet board providing this DeviceNet node and the board's firmware release number

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.3 Common I/O Data Transfer

dnetRead( )

Name: **dnetRead( )** - Read input data from a DeviceNet module

Synopsis:

```
STATUS    dnetRead
          (
HANDLE    handle,
int       MACID,
int       offset,
int       *rlen,
void      *buffer
          )
```

Description: This function reads input data from firmware's local data image of the DeviceNet module with *MACID*. Specify the same MACID as in the ***dnetStart( )*** call to access the input data of the local slave interface.

Parameter:

handle:	specifies the network to operate on
MACID:	MACID of the remote slave or local slave interface (0...63)
offset:	position of the data inside the stream of this device
**rlen*:	in: # of bytes to read or max. *buffer* space, respectively out:# of bytes really transferred
**buffer*:	points to a buffer to store data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetWrite( )

Name: **dnetWrite( )** - Write output data to a DeviceNet module

Synopsis:

```
STATUS   dnetWrite
         (
HANDLE   handle,
int      MACID,
int      offset,
int      len,
void     *buffer
)
```

Description: This function writes output data to the firmware's local data image of the DeviceNet module with *MACID*. This new data is then sent immediately to a remote slave. Specify the same MACID as in the ***dnetStart( )*** call to access the output data of the local slave interface.

Parameter:

handle:	specifies the network to operate on
MACID:	MACID of the remote slave or local slave interface (0...63)
offset:	position of the data inside the stream of this device
len:	length of data to write
**buffer*:	points to buffer with the data to write

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.4 Special: BitStrobe From Slave to Scanner

dnetWriteBitStrobe( )

Name: **dnetWriteBitStrobe( )** -Write bit strobe data to the DeviceNet scanner

Synopsis:

```
STATUS      dnetWriteBitStrobe
            (
            HANDLE      handle,
            int         flags,
            void        *buffer
            )
```

Description: This function writes 8 bytes of data to the scanner that are used as bit strobe data for any remote slave configured with a bit strobe connection. Or you may configure the data length of the bitstrobe frame to zero or eight bytes. In any case *buffer* must point to a 8 byte memory space.

Parameter:

handle: specifies the network to operate on

flags: Windows:

flags are reserved, please preset to (0x08) for compatibility with future releases.

VxWorks:

mask 0x08/0x00 selects 8 or 0 bytes bitstrobe frame, mask 0x01/0x00 selects immediate update or in next bitstrobe cycle.

**buffer*: pointer to a buffer for the data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetReadBitStrobe( )

Name: **dnetReadBitStrobe( )** - Read data from the bit strobe connection

Synopsis:

```
STATUS    dnetReadBitStrobe
          (
          HANDLE      handle,
          int         MACID,
          void        *buffer
          )
```

Description: If the local slave is scanned by a remote scanner via a bitstrobe connection then you can read the data received from the bitstrobe connection with this function. You always get eight bytes of data.

Parameter:

handle: specifies the network to operate on
MACID: MACID of the local slave interface (0...63)
**buffer*: pointer to a buffer for the data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.5 Direct Data/Explicit Transfer via Scanner

dnetWriteRead( )

Name: **dnetWriteRead( )** - Write data and read answer

Synopsis:

```
STATUS    dnetWriteRead
          (
          HANDLE handle,
          int    MACID,
          int    tlen,
          int    *rlen,
          void   *buffer
          )
```

Description: This function writes the data to a remote DeviceNet slave and reads after reception the answer.

Parameter:

handle:	specifies the network to operate on
MACID:	MACID of the remote slave (0...63)
tlen:	length of data to be transmitted
**rlen*:	max. length to receive, returns received length
**buffer*:	pointer to buffer to store data (Tx and Rx)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetExplRequest( )

Name: **dnetExplRequest( )** - Send an explicit request and read response

Synopsis:

```
STATUS    dnetExplRequest
          (
HANDLE    handle,
int       MACID,
int       service,
int       dnClass,
int       inst,
int       attr,
int       tlen,
int       *rlen,
void      *buffer
          )
```

Description: This function sends an explicit request to a remote module and returns the response after the reception.

Parameter:

handle:	specifies the network to operate on
MACID:	MACID of the remote module (0...63)
service:	DeviceNet service
dnClass:	DeviceNet class ID
inst:	DeviceNet instance ID
attr:	DeviceNet attribute ID (if zero, no attribute ID)
tlen:	length of data to be transmitted
**rlen*:	max. length to receive, returns received length
**buffer*:	points to buffer to store data (Tx and Rx) The buffer holds after return the explicit message body of the response, means the first byte is the service&response byte. For the format of the explicit message body please refer to the DeviceNet specification.

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.6 State Information

dnetReadFail( )

Name: **dnetReadFail( )** - Read status information of the DeviceNet interface

Synopsis:

```
STATUS    dnetReadFail
          (
HANDLE    handle,
void      *buffer
          )
```

Description: This function reads status information of the DeviceNet interface. The function reads a 64 bit = 8 byte array. Each set bit represents a failed or absent module (bit position = *MACID*), but bits of not installed modules are cleared.

The LSB of buffer [0] is bit 0 [-> MACID0].
The MSB of buffer [7] is bit 63 [-> MACID63].

Parameter:

handle:	specifies the network to operate on
**buffer*:	points to buffer to store data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetReadState( )

Name: **dnetReadState( )** - Read status of the DeviceNet module

Synopsis:

```
STATUS    dnetReadState
          (
HANDLE    handle,
int       MACID,
int       *state
          )
```

Description: This function reads the status of DeviceNet module with *MACID*. Specify the same MACID as in the ***dnetStart( )*** call to access the status of the local slave interface.

Parameter:

handle: specifies the network to operate on
MACID: MACID of the remote module or local slave interface (0...63)
**state*: variable that returns the state of the selected module as described below

States of Remote/Internal Slaves	
0	STATE_NOT_EXISTENT
1	STATE_WAIT_CONNECT
2	STATE_CONFIGURING
3	STATE_OPERATIONAL
4	STATE_TIMEOUT
5	STATE_ERROR

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.7 Event Handling

dnetEvEnable( )

Name: **dnetEvEnable( )** - Enable events for a network and a handle

Synopsis:

```
STATUS    dnetEvEnable
          (
          HANDLE          handle
          )
```

Description: This function enables the possibility to read events on the handle *handle*. Event generation is enabled for the network *handle* is opened for.

Parameter: *handle*: specifies the network to operate on

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

NOTICE You must NOT use a handle with events enabled for other purposes than reading events. This means it can only be used for ***dnetEvRead( )*** or ***dnetEvDisable( )***.

Header: `dnet.h`

dnetEvDisable( )

Name: **dnetEvDisable( )** - Disable events for a network and a handle

Synopsis:

```
STATUS    dnetEvDisable
          (
          HANDLE          handle
          )
```

Description: This function stops event generation for the net *handle* is valid for.
The last generated event on this net is:
EventNr == SERVICE_CLOSE
AddInfo == EVENT_SRV
This event may be used to exit from the event read loop.

Parameter: *handle*: specifies the network to operate on

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetEvRead( )

Name: **dnetEvRead( )** - Read one event of a DeviceNet network

Synopsis:

```
STATUS    dnetEvRead
          (
HANDLE    handle,
int       *EventNr,
int       *AddInfo,
int       *MACID
)
```

Description: This function reads one event from the network specified by *handle*.

Parameter:

handle:	handle with events enabled by ***dnetEvOpen ( )***
**EventNr*:	returns event number
**AddInfo*:	returns additional information value
**MACID*:	returns the MACID of the DeviceNet module that triggered the event

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.8 Host as Server for Explicit Messages

dnetExplSrvEnable( )

Name: **dnetExplSrvEnable( )** - Enable transfers between remote scanner and local slave

Synopsis:

```
STATUS    dnetExplSrvEnable
          (
          HANDLE    handle,
          int       MACID
          )
```

Description: This function enables the direct exchange of explicit messages between a remote scanner and the local slave through the vendor specific class 100. After this call explicit requests of a remote scanner are forwarded to the local application.

Parameter:
handle: handle of the specified net
MACID: MACID of the local slave interface (0...63)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetExplSrvDisable( )

Name: **dnetExplSrvDisable( )** - Disable transfers between remote scanner and local slave

Synopsis:

```
STATUS    dnetExplSrvDisable
          (
          HANDLE    handle,
          int       MACID
          )
```

Description: This function disables the forwarding of explicit messages from a remote scanner to the local slave and then to the local application program.

Parameter:

handle: handle returned by ***dnetOpen ( )***
MACID: MACID of the internal slave interface (0...63)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetExplSrvRead( )

Name: **dnetExplSrvRead( )** - Wait for a request from remote client and return its service request type

Synopsis:

```
STATUS    dnetExplSrvRead
          (
          HANDLE      handle,
          int         *cnxn,
          int         *service,
          int         *dnClass,
          int         *inst,
          int         *attr,
          int         *rlen,
          void        *buffer
          )
```

Description: This function waits for an explicit request from a remote client and returns it.

Parameter:

handle:	handle on which ***dnetExplSrvEnable ( )*** was done
**cnxn*:	connection number, return to card on response
**service*:	service number, return to card on response
**dnClass*:	client request to class
**inst*:	client request to instance
**attr*:	client request to attribute
**rlen*:	size of buffer space, returns received data length
**buffer*:	pointer to buffer to store data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetExplSrvResponse( )

Name: **dnetExplSrvResponse( )** - Response on request from remote client

Synopsis:

```
STATUS    dnetExplSrvResponse
          (
          HANDLE      handle,
          int         cnxn,
          int         service,
          int         len,
          void        *buffer
          )
```

Description: ***dnetExplSrvResponse ( )*** is the response of an explicit request from a remote client.

Parameter:

handle:	handle returned by ***dnetOpen ( )***
cnxn:	connection number, returned to card on response
service:	service number, returned to card on response
len:	size of buffer space, returns received data length
**buffer*:	pointer to buffer of data to transmit

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

2.9 Host as Server for Polled I/O-Connections

dnetPollSrvEnable( )

Name: **dnetPollSrvEnable ( ) -** Enable I/O-transfers between remote scanner and local slave

Synopsis:

```
STATUS    dnetPollSrvEnable
          (
          HANDLE          handle,
          int             MACID
          )
```

Description: This function enables the direct reception and reply of I/O-messages from a remote scanner.

Parameter:
handle: opened by ***dnetOpen ( )***
MACID: MACID of the local slave interface (0...63)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetPollSrvDisable( )

Name: **dnetPollSrvDisable( ) -** Disable I/O-transfers between remote scanner and local slave

Synopsis:

```
STATUS    dnetPollSrvDisable
          (
          HANDLE          handle,
          int             MACID
          )
```

Description: This function disables the direct exchange of I/O-polling data between a remote scanner and the local slave.

Parameter:
handle: handle returned by ***dnetOpen ( )***
MACID: MACID of the internal scanner and slave interface (0...63)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetPollSrvRead( )

Name: **dnetPollSrvRead( )** - Wait and read I/O-message from remote server

Synopsis:

```
STATUS    dnetPollSrvRead
          (
          HANDLE handle,
          int                MACID,
          int                *rlen,
          void               *buffer
          )
```

Description: This function waits for an I/O-message from a remote scanner and reads it.

Parameter:

handle:	handle on which ***dnetPollSrvEnable ( )*** was done
**rlen*:	size of buffer space, returns received data length
**buffer*:	pointer to buffer to store data
MACID:	MACID of the internal slave interface (0...63)

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

dnetPollSrvResp( )

Name: **dnetPollSrvResp( )** - Response I/O-poll request from remote scanner

Synopsis:

```
STATUS    dnetPollSrvResp
          (
HANDLE    handle,
int       MACID,
int       len,
void      *buffer
          )
```

Description: ***dnetPollSrvResp ( )*** is the response of an I/O-poll request of a remote scanner.

Parameter:

handle:	handle returned by ***dnetOpen ( )***
MACID:	MACID of the internal slave interface (0...63).
len:	size of data
**buffer*:	pointer to buffer to store data

Return: After successful execution `DNET_OK` is returned, otherwise an error code as described in chapter 3 is returned.

Header: `dnet.h`

3. Error Codes

Error code		Description
-1	DNET_INTERNAL_ERROR	other errors, e.g. DeviceNet scanner has stopped
0	DNET_OK	no error
1	DNET_WRONG_COMMAND	command not implemented
2	DNET_WRONG_PARA	general parameter error
3	DNET_WRONG_LENGTH	wrong length selected
4	DNET_WRONG_NET	wrong net-no selected
5	DNET_WRONG_MACID	wrong MACID
6	DNET_INV_DATA_ACCESS	data access out of range
7	DNET_WRONG_BAUDRATE	wrong bit rate selected
8	DNET_ALLREADY_EXIST	handle already exists
9	DNET_WRONG_STATE	wrong status
10	DNET_DISCONNECTED	status = disconnected
11	DNET_ON_TEST	-
12	DNET_NOT_STARTED	module not started
13	DNET_SRV_PENDING	service already pending

4. Events

The following events can occur:

Event	Event_no
NET_EVENT	1
TIMEOUT_EVENT	2
ERROR_EVENT	3
STATE_CHANGE	4
DATA_EVENT	5
CHANGE_EVENT	6
EXPLICIT_REQUEST	7
EXPLICIT_RESPONSE	8
POLL_REQUEST	9
POLL_RESPONSE	10
NET_REMOVED	11
SERVICE_CLOSE	12

Further additional information are returned with the events:

Additional Info	Info Code
BUS_OK_STATE	1
BUS_WARN_STATE	2
BUS_OFF_STATE	3
EXPLICIT_CNX	4
POLLING_CNX	5
BIT_STROBE_CNX	6
COS_CYCLIC_CNX	7
DUPLICATE_MACID	8
CONFIGURATION	9
FRAG_RESPONSE	10
ABORTED	11
PROT_ERROR	12
ERROR_ACKN	13
NO_RESOURCE	14
TIME_OUT	15
EVENT_SRV	16
EXPL_SRV	17
POLL_SRV	18

5. Using the DeviceNet Software

5.1 How to Write or Fit an Application

1. Open a handle using ***dnetOpen ( )***

2. Start the interface using ***dnetStart ( )***

> **NOTICE**
> The MACID is freely selectable, but must be unique inside the network (duplicate MACID check).

The *rxlen* and *txlen* means the consumed and produced data length of the I/O-connection to the internal slave interface. If you do not want to build this I/O-connection, set these parameters to zero.

3. Define the remote slaves. The scanner should access ***dnetInstRemSlave ( )***

> **NOTICE**
> The scanner checks *rxlen* and *txlen* against a found remote slave. If this check fails, the connection is not established! So you have to define the exact data length of the I/O-polling connection.

After this command the scanner tries immediately to connect this slave.

4. Check successful connection using ***dnetReadFail ( )***
Every not established or failed connection is shown by this command.

5. Read/Write data using ***dnetRead ( )*** and
dnetWrite ( )

6. Stop the application using ***dnetClose ( )***
This command closes only the handle opened by ***dnetOpen ( )***.The DeviceNet interface is not affected. A following application can directly access the DeviceNet data after ***dnetOpen ( )***.

5.2 Access Example

You have connect a flex I/O-module (AB) with a single digital input module IB16 (16-bit input) to your network.

The produced data length is 4 (= *rxlen* in ***dnetInstRemSlave ( )***):
2 bytes state information from the flex I/O-adapter and 2 bytes data from the input module.

The consumed data length is 2 (= *txlen* in ***dnetInstRemSlave ( )***):
2 bytes configuration for the input module.

If you want to read the input-data word by ***dnetRead ( )***, the resulting offset is 2 (data word after state information) and the length is 2 (word).

6. DeviceNet Errata DN-PCI/331

The DN-PCI/331 board has not been presented to an official ODVA test lab to apply for a conformance certificate. However the board's protocol implementation of the DeviceNet slave has been checked with the conformance test tool (CTT) that official ODVA test labs use while doing the conformance test for a DeviceNet appliance. The used release of the CTT was “DeviceNet Protocol Conformance Revision A22, November 2010”.
During the tests some deviating behaviour was observed that is described in the following paragraphs:

6.1 Master /Scanner: The UCMM loses Message IDs

The scanner needs to allocate a message ID for the UCMM connection when it starts talking to an external slave. If the external slave does not answer and in that state the connection is closed using *dnetInstRemSlave(allocation=0)* then the UCMM Connection ID may be locked and the connection ID is lost for later connection attempts. There is only the limited amount of 16 connection IDs available. This is a severe problem if you try to scan the DeviceNet network for any slaves present and you possibly lose a connection ID on every slave not present.
It is only a minor problem if you only need to talk to a fixed list of slaves and do not use *dnetInstRemSlave(allocation=0)* because as soon as a slave answers, its connection ID is returned to the pool and reused.
This error does only affect the master/scanner side and is not tested in ODVA's conformance test.

6.2 Master/Scanner: CoS Allocation is incorrect

When using the CoS connection only, conflicts with the required behaviour of the slave occur. In particular a single CoS connection can not be chosen.
The allocation byte is determined by the configuration of the external DeviceNet slave via the function *dnetInstRemSlave()*, see table 2.1.1 on page 11.
The behaviour for the allocation of a CoS/Cyclic connection together with a Polled connection is described for the slave side in the DeviceNet specification Volume 3, Chapter 3.13 "Change of State/Cyclic Messages". In particular the behaviour, in the case that both connections (CoS/Cyclic & Polled) are not enabled using a single allocation request, is precisely specified.

The following deviations from the specification occur:

- If you use an allocation byte of 0x11 (CoS only & Explicit) in *dnetInstRemSlave()*, the scanner will implicitly switch on the Polled connection if the consumed length of the external slave is unequal zero (0). According to the standard, the scanner should only use the CoS connection and leave the Polled connection unused. Therefore the board's scanner can not use the CoS connection as an input only connection.
- If you use an allocation byte of 0 x13 (CoS & Polled & Explicit) in *dnetInstRemSlave()*, the scanner will wrongly send an allocation request with a choice byte of 0x01 (Explicit) and then 0x02 (Polled Only).

NOTICE
To avoid these errors do not use the allocation bytes 0x11 (CoS only & Explicit) or 0x13 (CoS & Polled & Explicit) when calling *dnetInstRemSlave()* for an external slave.

6.3 Slave/Device: Network Status is not always indicated correctly

During the conformance test the network status is not always indicated correctly by the Status LED (especially if only UCMM connections do exist).
The DeviceNet object test of the CTT performs a Network Status LED test, because a combined Module/Network Status LED is provided. The Network Status LED test applies also to the combined LED, because the CTT does only test the interaction on the network. The states of the hardware, which are to be additionally indicated by the combined LED, do not occur during the test and consequently do not affect the display of the combined LED.
When the state machine is run though, the blinking codes of the combined DeviceNet Module/Network Status LED do not indicate the DeviceNet network states correctly.
For example, the slave should switch the LED to light permanently green already if a connection is dynamically established via UCMM, but the LED remains blinking green.

6.4 Network Access State Machine after Communication Failure

This problem occurs in the communication failure state (for example after a CAN bus-off). When the 24V power supply voltage of the DeviceNet interface is removed and then supplied again in that state, the Network Access State Machine should restart, carry out a Duplicate MACID Check and leave the communication failure state on Duplicate MACID Check success. But this does not happen.

INFORMATION
To leave the communication failure state you may power down the DN-PCI/331 or reconfigure all slaves and then call *dnetStop()* followed by *dnetStart()*. This should reset this condition.

6.5 Network Status on DeviceNet Power Loss

The network status at loss of the 24V DeviceNet power supply voltage is not indicated correctly.
The LED is not switched to blinking red if the power supply voltage on the DeviceNet bus is not supplied.

6.6 Restriction in the EDS File for the DeviceNet Slave Part of the Device

The EDS file currently does not reflect, that the I/O lengths of the DeviceNet slave are completely configurable. The EDS itself declares only fixed input and output lengths for the slave part of the device.

INFORMATION
To work around this problem enter values of your configuration for the amount of input and output bytes into the EDS file of the DN-PCI/331 manually at the appropriate places.