CAN/Can Open Driver Specification

CAN / CANOpen CiA 301 Specification

Contents

General Information

This chapter identifies all of the hardware and software components required to implement communication between the CAN driver in Studio and CAN/CANOpen devices.

The information is organized into the following sections:

- Device Specifications
- Network Specifications
- Driver Characteristics

Device Specifications

To establish communication, your target device must meet the following specifications:

- **Manufacturer**: Peak, Advantech, Hilscher
- **CAN Compatible Equipment**:
  - Hilscher PCI Board CIF50
  - Peak-USB and Peak PCI devices
  - TPC-660G Delta, Delta-CPU2
  - TPC-120H

CAN messages has been tested successfully with Advantech TPC-660G, Peak-USB, Peak-PCI and Hilscher PCI Board CIF-50, whereas CANOpen has been tested using FnIO S–Series: KNA9161.

Network Specifications

To establish communication, your device network must meet the following specifications:

- **Device Communication Port**: Selectable
- **Physical Protocol**: CAN Cia Standard 102
- **Logic Protocol**:
  - CAN Cia Standard 201
  - CANOpen Cia Standard 301

Driver Characteristics

The CAN driver package consists of the following files, which are automatically installed in the **`/DRV`** subdirectory of Studio:

- **`CAN.INI:`** Internal driver file. *You must not modify this file*.
- **`CAN.MSG:`** Internal driver file containing error messages for each error code. *You must not modify this file*.
- **`CAN.PDF:`** This document, which provides detailed information about the CAN driver.
- **`CAN.DLL:`** Compiled driver
- **`CIF32DLL.DLL:`** Hilscher API. It must be placed on the **`/DRV/API`** subdirectory of Studio. This file is provided by Hilscher
- **`PCANBasic.DLL:`** Peak API. It must be placed on the **`/DRV/API`** subdirectory of Studio. This file is provided by Peak

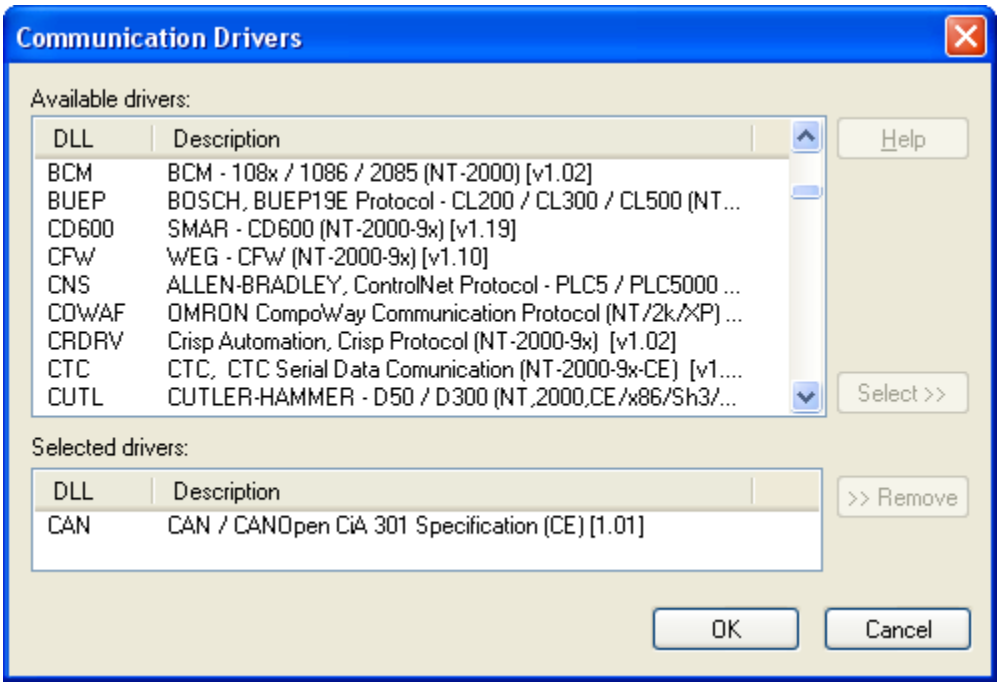
Notes:

- **Driver Runtime APIs**: CIF32DLL.dll API provided by Hilscher or PCANBasic.dll provided by Peak. Advantech does not require an API. Please, install peak and Hilscher drivers and copy CIF32DLL.dll or PCANBasic.dll in folder `\DRV\API\` subfolder

Selecting the Driver

When you install Studio, all of the communication drivers are automatically installed in the **\DRV** subdirectory but they remain dormant until manually selected for specific applications. To select the CAN driver for your Studio application:

1. From the main menu bar, select **Insert** → **Driver** to open the *Communication Drivers* dialog.
2. Select the **CAN** driver from the *Available Drivers* list, and then click the **Select** button.



Communication Drivers Dialog

3. When the **CAN** driver is displayed in the **Selected Drivers** list, click the **OK** button to close the dialog. The driver is added to the *Drivers* folder, in the *Comm.* tab of the Workspace.

Attention:

For safety reasons, you must use special precautions when installing the physical hardware. Consult the hardware manufacturer's documentation for specific instructions in this area.

Configuring the Device

Consult your CAN documentation for information about configuring your device.

Configuring the Driver

Once you have selected the CAN driver in Studio, you must properly configure it to communicate with your target device. First, you must set the driver's communication settings to match the parameters set on the device. Then, you must build driver worksheets to associate database tags in your Studio application with the appropriate addresses (registers) on the device.

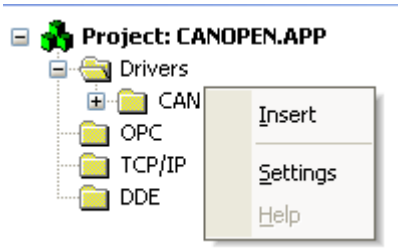
Configuring the Communication Settings

The communication settings are described in detail in the "Communication" chapter of the Studio *Technical Reference Manual*, and the same general procedures are used for all drivers. Please review those procedures before continuing.

For the purposes of this document, only CAN driver-specific settings and procedures will be discussed here. To configure the communication settings for the CAN driver:

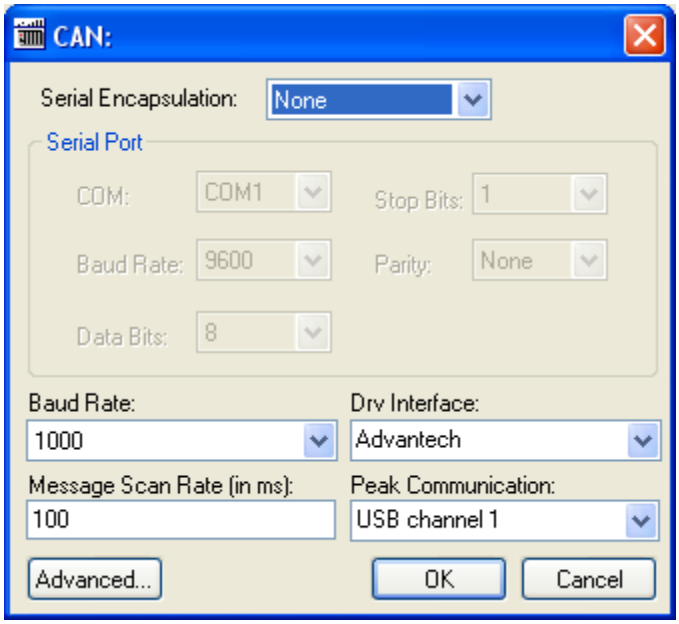
1. In the *Workspace* pane, select the *Comm.* tab and then expand the *Drivers* folder. The CAN driver is listed here as a subfolder.

2. Right-click on the *CAN* subfolder and then select the **Settings** option from the pop-up menu:



Select Settings from the Pop-Up Menu

The *CAN: Communication Settings* dialog is displayed:



CAN: Communication Settings Dialog

3. In the *Communication Settings* dialog, configure the driver settings to enable communication with your target device. To ensure error-free communication, the driver settings must *exactly match* the corresponding settings on the device. Please consult the manufacturer's documentation for instructions how to configure the device and for complete descriptions of the settings.

   Depending on your circumstances, you may need to configure the driver *before* you have configured your target device. If this is the case, then take note of the driver settings and have them ready when you later configure the device.

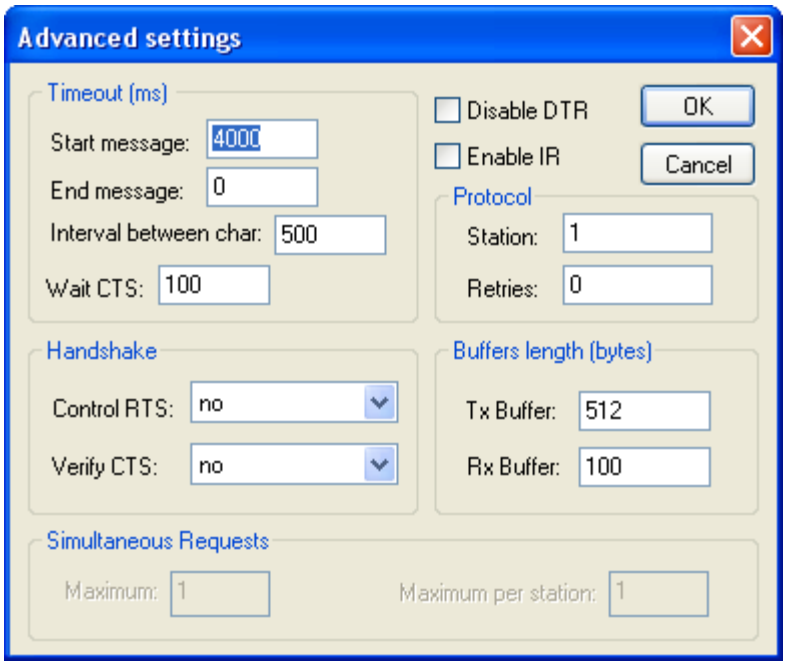
> ➲ **Attention:**
> For safety reasons, you **must** take special precautions when connecting and configuring new equipment. Please consult the manufacturer's documentation for specific instructions.

The communication settings and their possible values are described in the following table:

Parameters	Default Values	Valid Values	Description
Baud Rate (kbps)	1000	1000, 800, 500, 250,125, 100, 50, 20, 10	Defines the communication baud rate. OBS: Both Peak and Advantech TPC-660G interface does not support 800kbps baud rate
Message Scan Rate (ms)	100	Greater than 0	Defines the minimum time that the driver must wait before checking the CAN bus for messages.
DRV Interface		Peak,	Configures the manufacturer of CAN layer

		Hilscher, TPC (Advantech)	
Peak Communication Type	USB channel 1	USB channel 1 USB channel 2 USB channel 3 USB channel 4 USB channel 5 USB channel 6 USB channel 7 USB channel 8 PCI channel 1 PCI channel 2 PCI channel 3 PCI channel 4 PCI channel 5 PCI channel 6 PCI channel 7 PCI channel 8 ISA channel 1 ISA channel 2 ISA channel 3 ISA channel 4 ISA channel 5 ISA channel 6 ISA channel 7 ISA channel 8 DNGBUS PCCBUS channel 1 PCCBUS channel 2	Defines Peak communication interface. For others CAN implementations (Hilscher or Advantech) this parameter is ignored

4. Click on the **Advanced…** button in the *Communication Parameters* dialog. The *Advanced settings* dialog will display.



Parameters	Syntax	Default Values	Valid Values	Description
Rx Buffer	Size	512	Above 4	Buffer size used for CANOpen communication

Configuring the Driver Worksheets

Each selected driver includes a Main Driver Sheet and one or more Standard Driver Worksheets. The Main Driver Sheet is used to define tag/register associations and driver parameters that are in effect at all times, regardless of application behavior. In contrast, Standard Driver Worksheets can be inserted to define additional tag/register associations that are triggered by specific application behaviors.

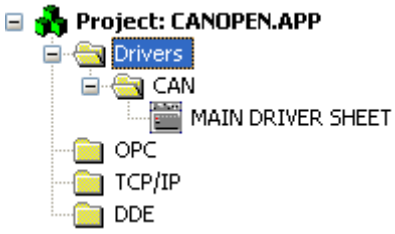
The configuration of these worksheets is described in detail in the "Communication" chapter of the Studio *Technical Reference Manual*, and the same general procedures are used for all drivers. Please review those procedures before continuing.

For the purposes of this document, only CAN driver-specific parameters and procedures are discussed here.

MAIN DRIVER SHEET

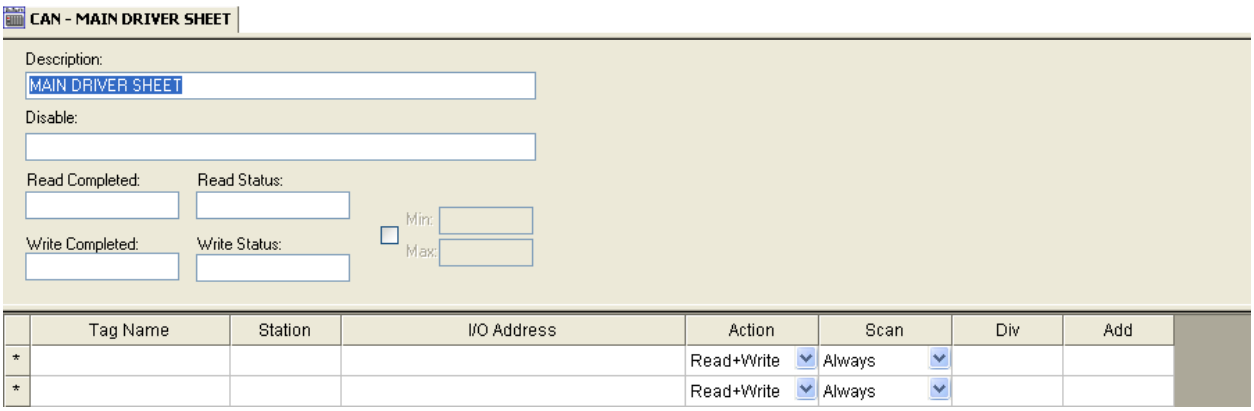
When you select the CAN driver and add it to your application, Studio automatically inserts the *Main Driver Sheet* in the *CAN* driver subfolder. To configure the Main Driver Sheet:

1. Select the *Comm.* tab in the *Workspace* pane.
2. Open the *Drivers* folder, and then open the *CAN* subfolder:



Main Driver Sheet in the CAN Subfolder

3. Double-click on the **MAIN DRIVER SHEET** icon to open the following worksheet:



Opening the Main Driver Sheet

Most of the fields on this sheet are standard for all drivers; see the "Communication" chapter of the *Technical Reference Manual* for more information on configuring these fields. However, the **Station** and **I/O Address** fields use syntax that is specific to the CAN driver.

4. For each table row (i.e., each tag/register association), configure the **Station** and **I/O Address** fields.

Use the **I/O Address** field to associate each tag to its respective device address and **Station** field to define the address of the device to be read from or written to.

The following sections describe how to configure **Station** and **I/O Address** for both CAN and CANOpen messages.

CAN Messages

The syntax below shows how to Read and Write in CAN net work. When the driver is Idle, the driver continuously gets messages from network and put them in an internal memory list. Reading behavior depends on Request and Synchronous options (as described below), but in most of cases reading is simply to pick up a message in list whose ID is the same one configured in Station Field. On the other hand, writing consists in update the internal memory list and to send a message whose ID is defined in station field.

Station Syntax

<CAN Message ID>: CAN Message ID (Hexadecimal).

I/O Address Syntax
CANMASTER.<Request Option>.<Synchronous Option > : <Address>
<Request Option>: REQNF, NOREQ, REQ, REQRESET
REQNF: Default behavior. When the driver is Idle, it continuously gets messages from network and put them in an internal list. Reading is simple picking the message from the list and places its values in the tags. If message is not in list, the driver sends a remote request.

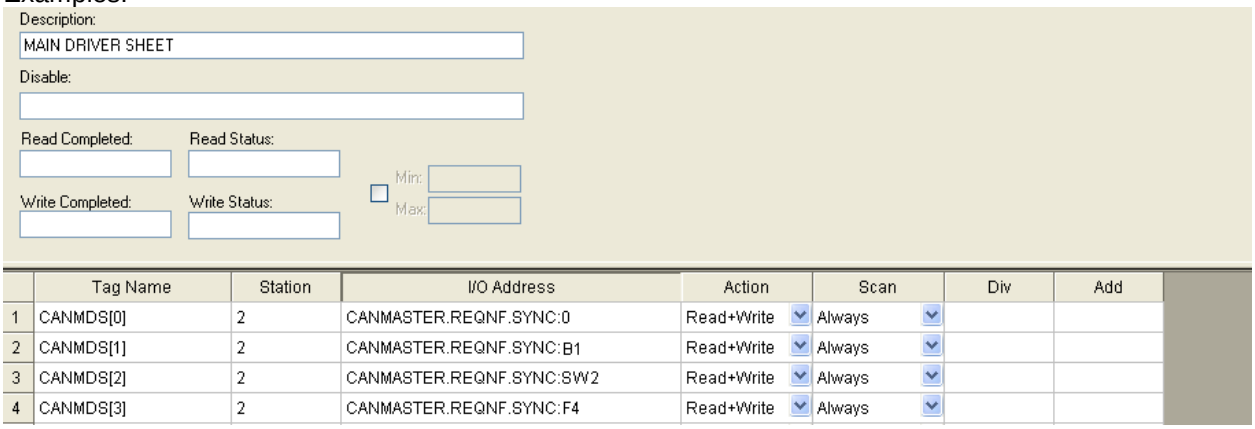
REQ : During reading, driver get the message from the list, but also sends a remote request, in order to always keep values updated

NOREQ: Does never perform a remote request. Only generates an error if message was not found

REQRESET: Delete the message from the list and performs a remote request.
<Synchronous Option >: SYNC, ASYNC

SYNC: Synchronous. Default behavior. If message is not in the list, the driver waits for the message, until a timeout.
ASYNC: Asynchronous. If message is not in the list, driver just generates an error. It may or not send a remote request according to Request Option.

Examples:



	Tag Name	Station	I/O Address	Action	Scan	Div	Add
1	CANMDS[0]	2	CANMASTER.REQNF.SYNC:0	Read+Write	Always		
2	CANMDS[1]	2	CANMASTER.REQNF.SYNC:B1	Read+Write	Always		
3	CANMDS[2]	2	CANMASTER.REQNF.SYNC:SW2	Read+Write	Always		
4	CANMDS[3]	2	CANMASTER.REQNF.SYNC:F4	Read+Write	Always		

Writing procedure consists in writing the message in CAN network. If all CAN eight bytes are defined, the driver just put the message in CAN network. On the other hand, if message is incomplete, driver must read it, according to criteria defined by Request Option and Synchronous Option.
<Address>: defines the CAN message data byte. CAN messages have at most eight data bytes, so, for default BYTE type, the address may range 0 to 7. However, you may add a prefix and change data type, as shown in table below. Moreover, some can messages can have fewer bytes. For this situation, make sure that the data does not exceed the message length.

Prefix	Data Type
B	Signed 8 bits variable (byte)
UB	Unsigned 8 bits variable (byte)
W	Signed 16 bits variable (word)
SW	Signed 16 bits variable (word) with byte swap
UW	Unsigned 16 bits variable (word)
USW	Unsigned 16 bits variable (word) with byte swap
DW	Signed 32 bits variable (double word)
SDW	Signed 32 bits variable (double word) with byte swap
UDW	Signed 32 bits variable (double word) with byte swap
USDW	Signed 32 bits variable (double word) with byte swap
UDW	Unsigned 32 bits variable (double word)
USDW	Unsigned 32 bits variable (double word) with byte swap
F	32 bits float points (float)
SF	32 bits float points with byte swap (float)
DF	64 bits float points (double)
SDF	64 bits float points with byte swap (double)
BCD	16 bits BCD value
BCDDW	32 bits BCD value
S	String (address should be S<register>.<Bytes>)

CANOpen Messages

In order to access to entries of a CANOPEN device Object Dictionary (OD), you should configure the Station and I/O Address field according to the syntax below. The driver communicates using SDO Downloading (Write) or SDO Uploading (Read). SDO Block Upload, SDO Block Upload and PDOs are not supported.

Station Syntax

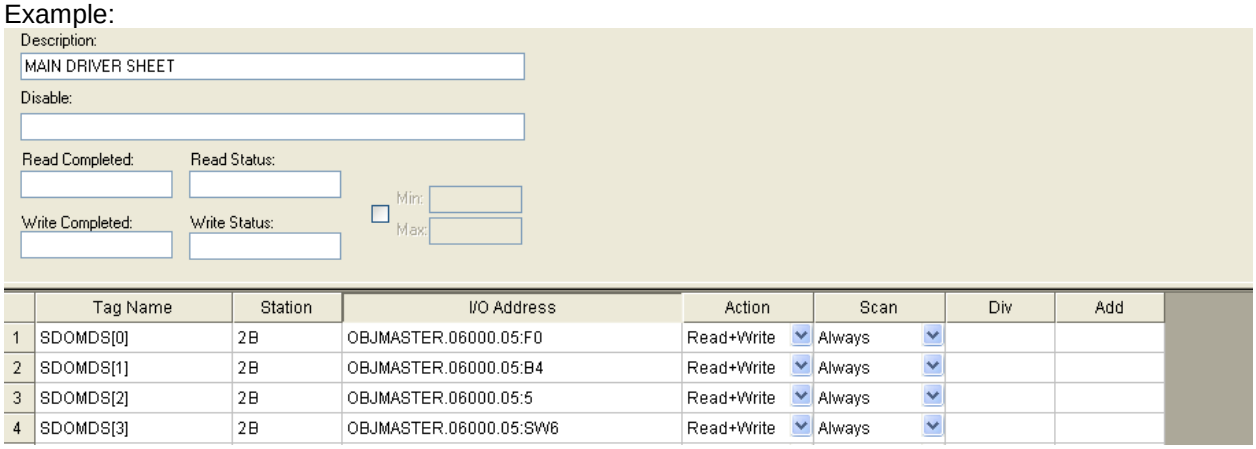
<Node ID>: Device node ID

I/O Address Syntax

OBJMASTER.<Index>.<Sub-Index> : < Address>

Where:
Index: Object Dictionary Index
Sub- Index: Object Dictionary Sub-Index
Address: Defines CANOpen data byte. As Object Dictionary entries may contain data of arbitrary size, address may range according to Register data size. All prefixes supported by CANMaster header is also supported by OBJMASTER.

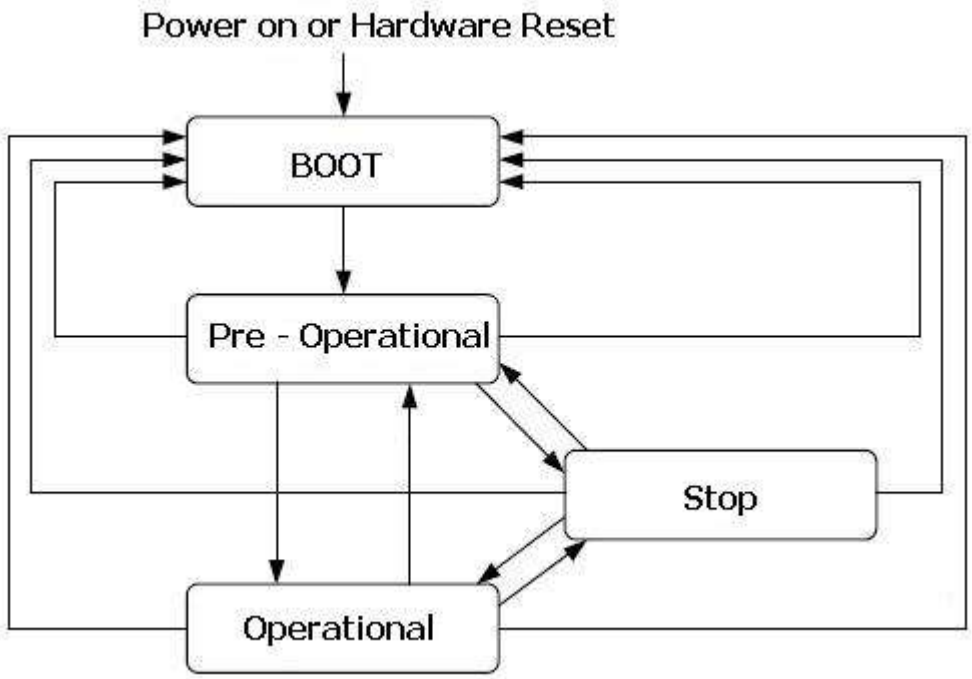
Example:



	Tag Name	Station	I/O Address	Action	Scan	Div	Add
1	SDOMDS[0]	2B	OBJMASTER.06000.05:F0	Read+Write	Always		
2	SDOMDS[1]	2B	OBJMASTER.06000.05:B4	Read+Write	Always		
3	SDOMDS[2]	2B	OBJMASTER.06000.05:5	Read+Write	Always		
4	SDOMDS[3]	2B	OBJMASTER.06000.05:SW6	Read+Write	Always		

CANOpen NMT Message

NMT are control messages, which are used to get or to set a CANOpen device state. According to Cia 301 Specification, there CANOpen PLC has four states: BOOT, PRE-OPERATIONAL, OPERATIONAL and STOP



I/O Address Syntax

NMT: < Address>

<Address>: Device node ID. You should use string type and size should be greater or equal than 9.

S<Node-ID>.9

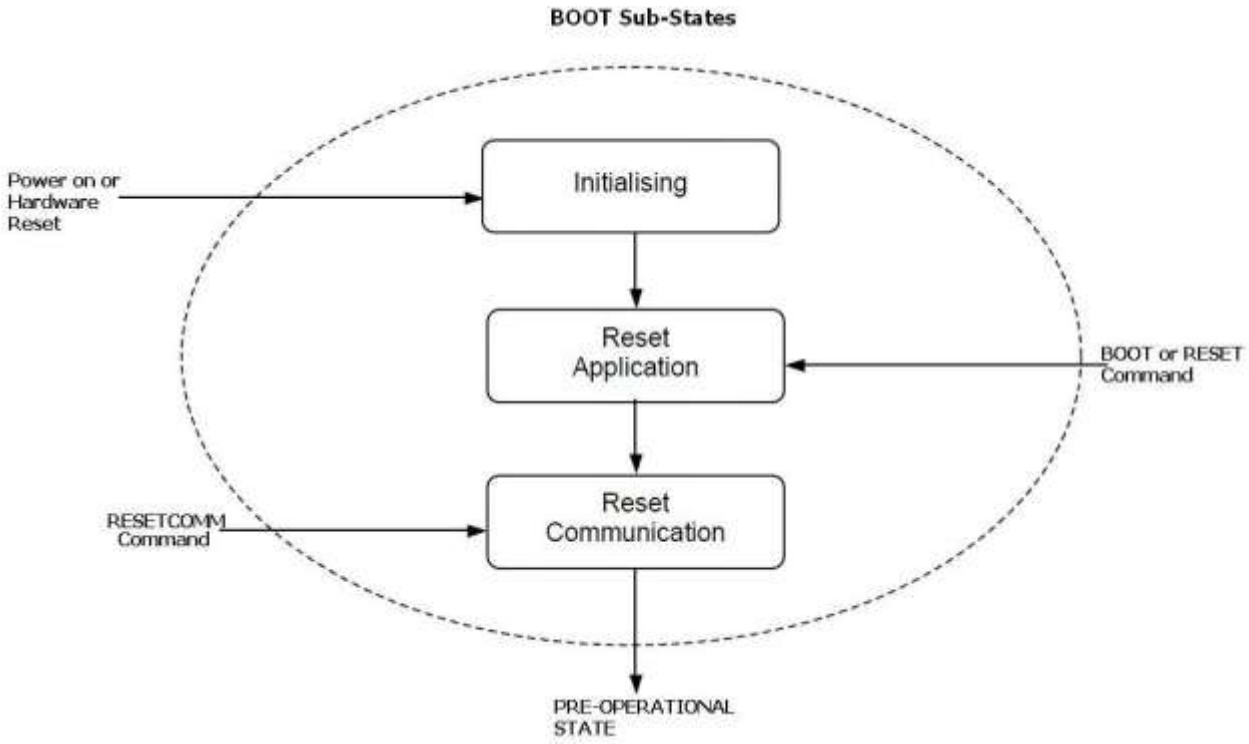
Station field should be kept empty.

Tags must be strings. If you read a device state tags will assume one of the four values: “OPER”, “PRE-OPER”, “BOOT”, “STOP”, as shown in table below. For writing, you should enter one of these four values. For state Boot and Operational, you can also enter values Reset or Start, respectively, as shown in table below.

Write Command	Read Value	Get / Set State
BOOT / RESET	BOOT	Boot
STOP	STOP	Stop
PREOP	PREOP	Pre operational
OPER/START	OPER	Operational

RESETCOMM	BOOT	Reset communication

The “initialization” or “boot” state is divided into three sub-states in order to enable a complete or partial reset of a node.

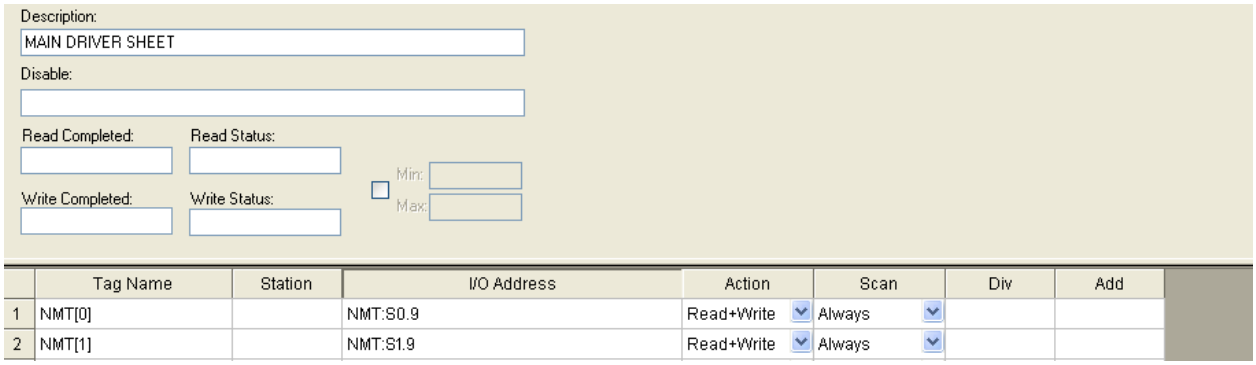


1. Initializing: This is the first sub-state the device enters after power-on or hardware reset. After finishing the basic node initialization the device enters autonomously into the state Reset_Application.

2. Reset_Application: In this state the parameters of the manufacturer specific profile area and of the standardized device profile area are set to their power-on values. After setting of the power-on values the state Reset_Communication is entered autonomously. Writing “RESET” or “BOOT”, device is enter in this sub-state

3. Reset_Communication: In this state the parameters of the communication profile area are set to their power-on values. After this the state Initialization is finished and the device executes the write boot-up object service and enters the state PRE-OPERATIONAL. Writing “RESETCOMM”, device enter in this substate

Example:



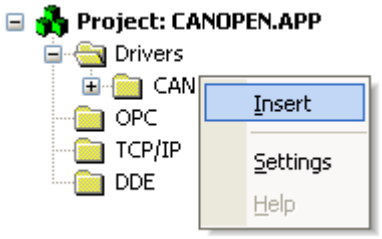
STANDARD DRIVER WORKSHEET

When you select the CAN driver and add it to your application, it has only a Main Driver Sheet by default (see previous section). However, you may insert additional Standard Driver Worksheets to define tag/register associations that are triggered by specific application behaviors. Doing this will optimize communication and improve system performance by ensuring that tags/registers are scanned only when necessary – that is, only when the application is performing an action that requires reading or writing to those specific tags/registers.

> **Note:**
> We recommend configuring device registers in sequential blocks in order to maximize performance.

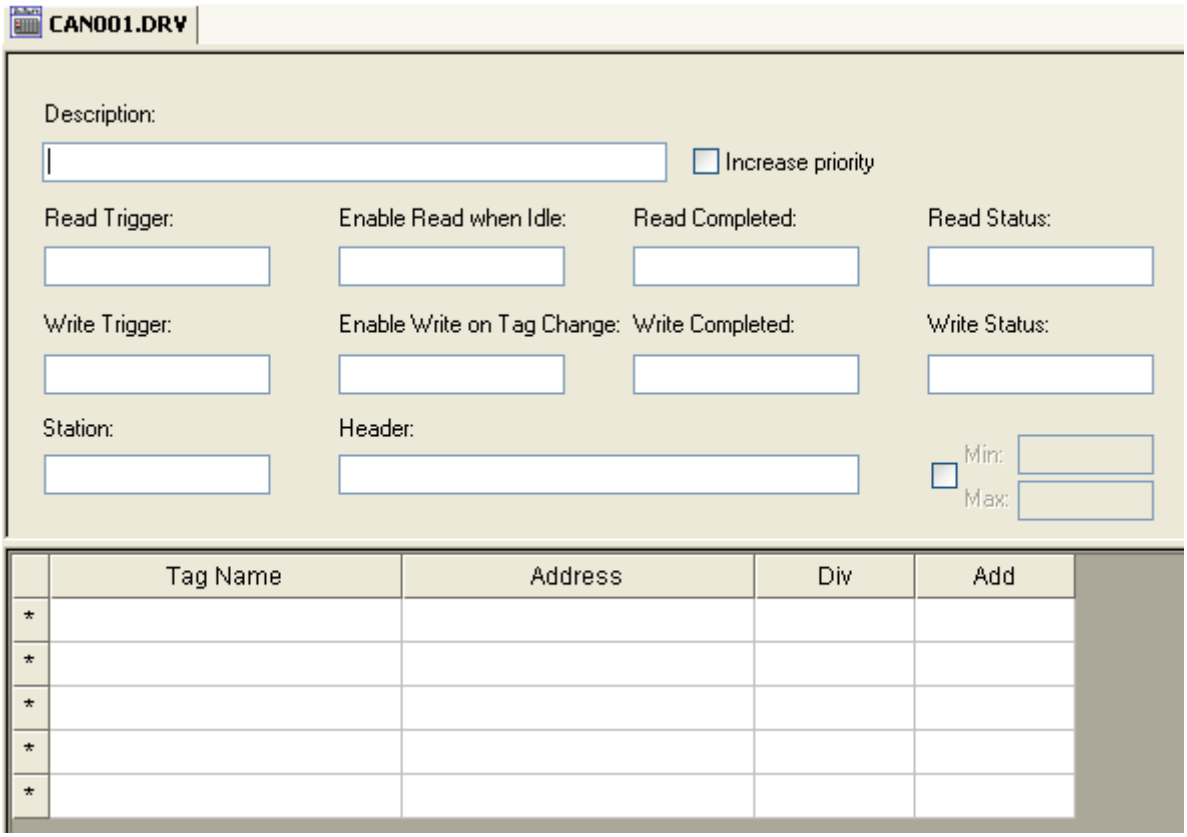
To insert a new Standard Driver Worksheet:

1. In the *Comm.* tab, open the *Drivers* folder and locate the *CAN* subfolder.
2. Right-click on the *CAN* subfolder, and then select **Insert** from the pop-up menu:



Inserting a New Worksheet

A new CAN driver worksheet is inserted into the *CAN* subfolder, and the worksheet is open for configuration:



CAN Driver Worksheet

> **Note:**
> Worksheets are numbered in order of creation, so the first worksheet is **`CAN001.drv`**.

Most of the fields on this worksheet are standard for all drivers; see the "Communication" chapter of the *Technical Reference Manual* for more information on configuring these fields. However, the **Station**, **Header**, and **Address** fields use syntax that is specific to the CAN driver.

3. The following sections describe how to configure **Station** and **Header** for both CAN and CANOpen messages.

CAN Messages - MASTER

The syntax below shows how to Read and Write in CAN network. When the driver is Idle, the driver continuously gets messages from network and put them in an internal memory list. Reading behavior depends on Request and Synchronous options (as described below), but in most of cases reading is simply to pick up a message in list whose ID is the same one configured in Station Field. On the other hand, writing consists in update the internal memory list and to send a message whose ID is defined in station field.

Station Syntax

<CAN Message ID>: CAN Message ID (Hexadecimal).

Header Syntax
CANMASTER.<Request Option>.<Synchronous Option > : <Offset>
<Request Option>: REQNF, NOREQ, REQ, REQRESET.
REQNF: Default behavior. When the driver is Idle, it continuously gets messages from network and put them in an internal list. Reading is simple picking the message from the list and places its values in the tags. If message is not in list, the driver sends a remote request.

REQ : During reading, driver get the message from the list, but also sends a remote request, in order to always keep values updated

NOREQ: Does never perform a remote request. Only generates an error if message was not found

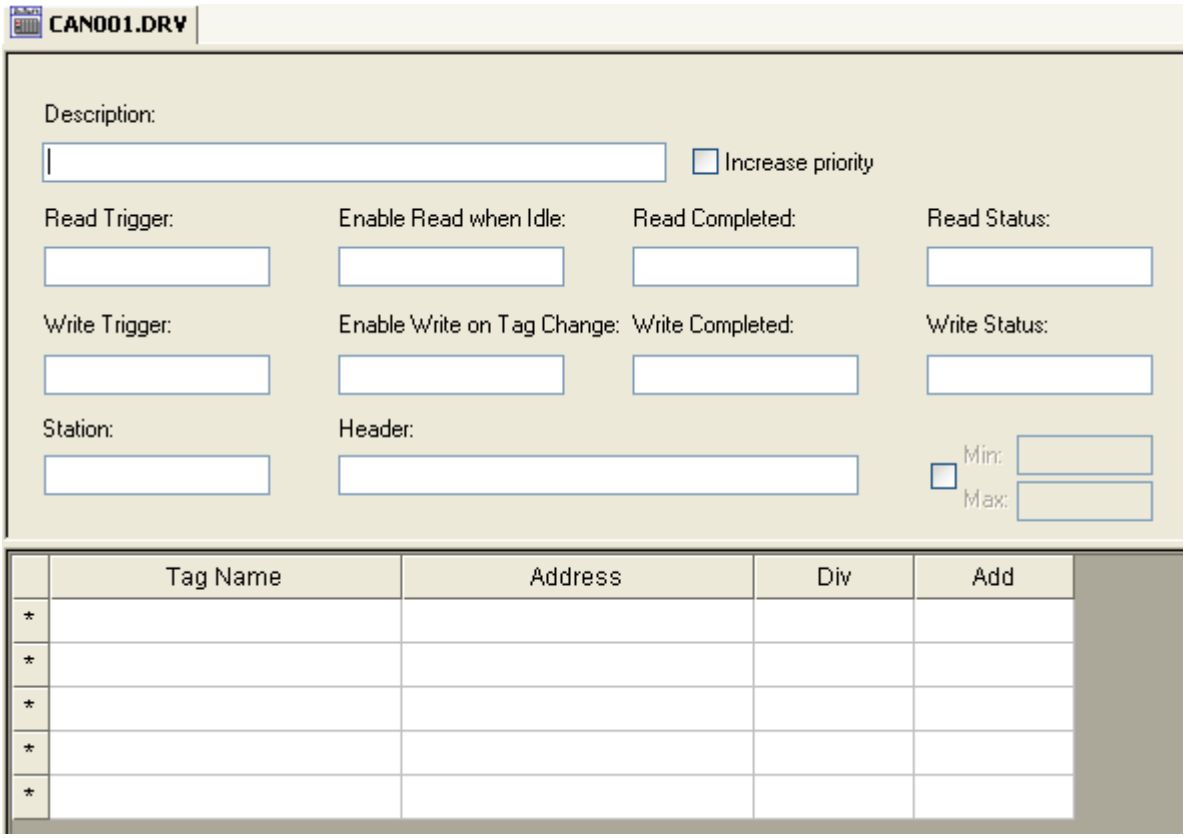
REQRESET: Delete the message from the list and performs a remote request.
<Synchronous Option >: SYNC, ASYNC

SYNC: Synchronous. Default behavior. If message is not in the list, the driver waits for the message, until a timeout.
ASYNC: Asynchronous. If message is not in the list, driver just generates an error. It may or not send a remote request according to Request Option.
<Offset>: Is the CAN message offset

Address: For each table row (i.e., each tag/register association), configure the **Address** field with the can message Data Byte. CAN messages have at most eight data bytes, so, for default BYTE type, the address may range 0 to 7. However, you may add a prefix and change data type, as shown in table below. Moreover, some can messages can have fewer bytes. For this situation, make sure that the data does not exceed the message length.

Prefix	Data Type
B	Signed 8 bits variable (byte)
UB	Unsigned 8 bits variable (byte)
W	Signed 16 bits variable (word)
SW	Signed 16 bits variable (word) with byte swap
UW	Unsigned 16 bits variable (word)
USW	Unsigned 16 bits variable (word) with byte swap
DW	Signed 32 bits variable (double word)
SDW	Signed 32 bits variable (double word) with byte swap
UDW	Signed 32 bits variable (double word) with byte swap
USDW	Signed 32 bits variable (double word) with byte swap
UDW	Unsigned 32 bits variable (double word)
USDW	Unsigned 32 bits variable (double word) with byte swap
F	32 bits float points (float)
SF	32 bits float points with byte swap (float)
DF	64 bits float points (double)
SDF	64 bits float points with byte swap (double)
BCD	16 bits BCD value
BCDDW	32 bits BCD value
S	String (address should be S<register>.<Bytes>)

Example



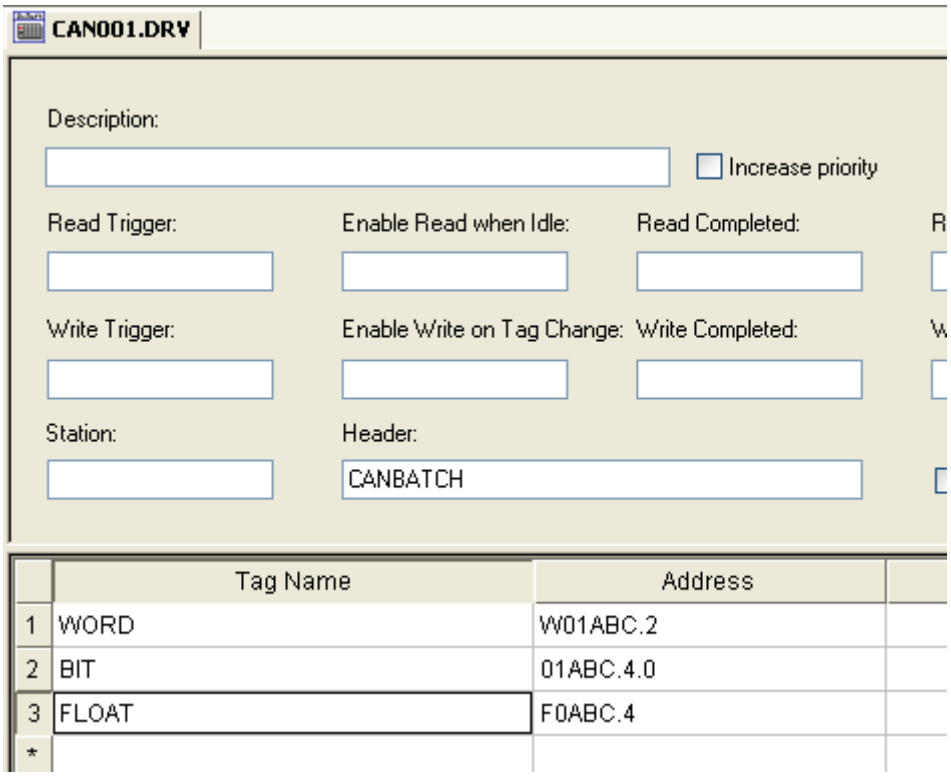
CAN Messages – BATCH

Using the header CANMASTER, the user is allowed to read one message at a time from the driver's internal list or request only a single message data. However, for better utilization of resources on performance limited architectures, the header CANBATCH allows to read several messages at once from the buffer of received messages. This header operates in a mode equivalent to **`CANMASTER.NOREQ.ASYNC`**, meaning it does not start remote requests for message IDs that were not found or waits for messages to arrive. The station field for this header must be left blank. The address field must be configured with a special syntax:

`<DataType><Message ID>.<Starting Byte>.<Bit Number>`

Where:
`<DataType>` is one of the data types of the table above (default is BYTE)
`<Message ID>` is the message identification, in hexadecimal format
`<Starting Byte>` is the first byte of the message to read using the <DataType>
`<Bit Number>` is the bit number to read from the desired byte (optional)

Examples:



Example of header CANBATCH

Attention:

The header CANBATCH is specifically designed for improving performance of read operations. It does not support write operations or usage on the Main Driver Sheet. Also, on the same driver, there must be at most one driver sheet using header CANBATCH. Otherwise, the behavior is undefined.

If a message ID is configured on one line of the driver sheet, and it was not received by the CAN driver, the value of its associated tag is zeroed and a message is printed on the log.

Changes on a driver sheet configured with header CANBATCH while the driver is running may not be considered until a restart of the driver.

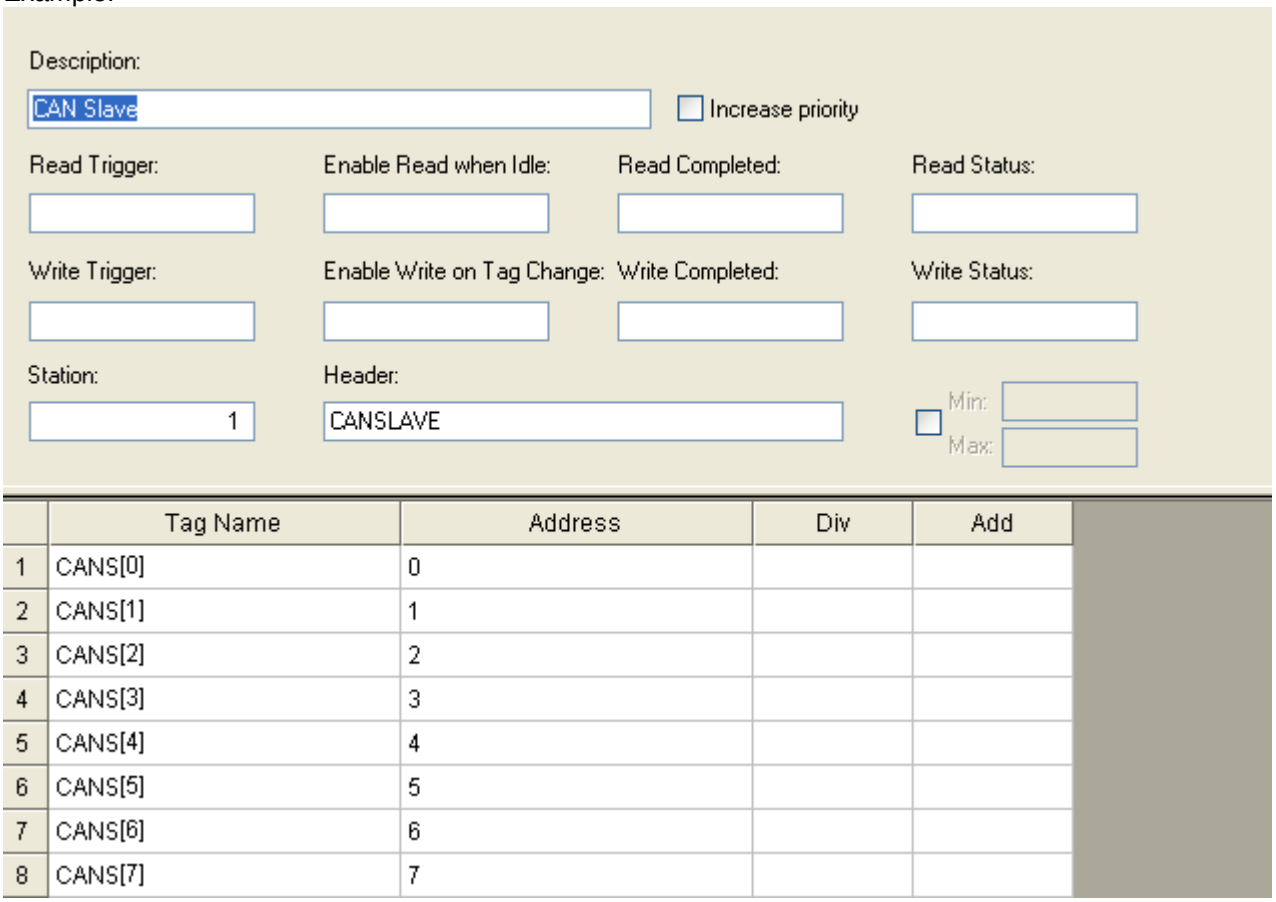
CAN Messages - SLAVE

The syntax below shows how configure a CAN slave device. This device may catch all CAN messages whose ID is the same one configured either in Settings->Advanced->Slave, or in the worksheet's *Station* field, and place its values into tags. The Slave can also answer for Master Remote Requests.

Header Syntax
CANSLAVE
Station
For CAN: defines the message ID (In Decimal notation) that will be used to receive a message sent by the CAN Master
For CANOpen: defines the node-ID of a simulated slave device

Address: For each table row (i.e., each tag/register association), configure the **Address** field with the can message Data Byte. CAN messages have at most eight data bytes, so, for default BYTE type, the address may range 0 to 7. However, you may add a prefix and change data type, as shown previously. Make sure that the data does not exceed the message 8 bytes length.

Example:



	Tag Name	Address	Div	Add
1	CANS[0]	0		
2	CANS[1]	1		
3	CANS[2]	2		
4	CANS[3]	3		
5	CANS[4]	4		
6	CANS[5]	5		
7	CANS[6]	6		
8	CANS[7]	7		

CANOpen Messages - MASTER

In order to access to entries of a CANOPEN device Object Dictionary (OD), you should configure the Station and I/O Address field according to the syntax below. SDO Block Upload, SDO Block Upload and PDOs are not supported.

Station

<Node ID>: Device node ID (HEX)

Header

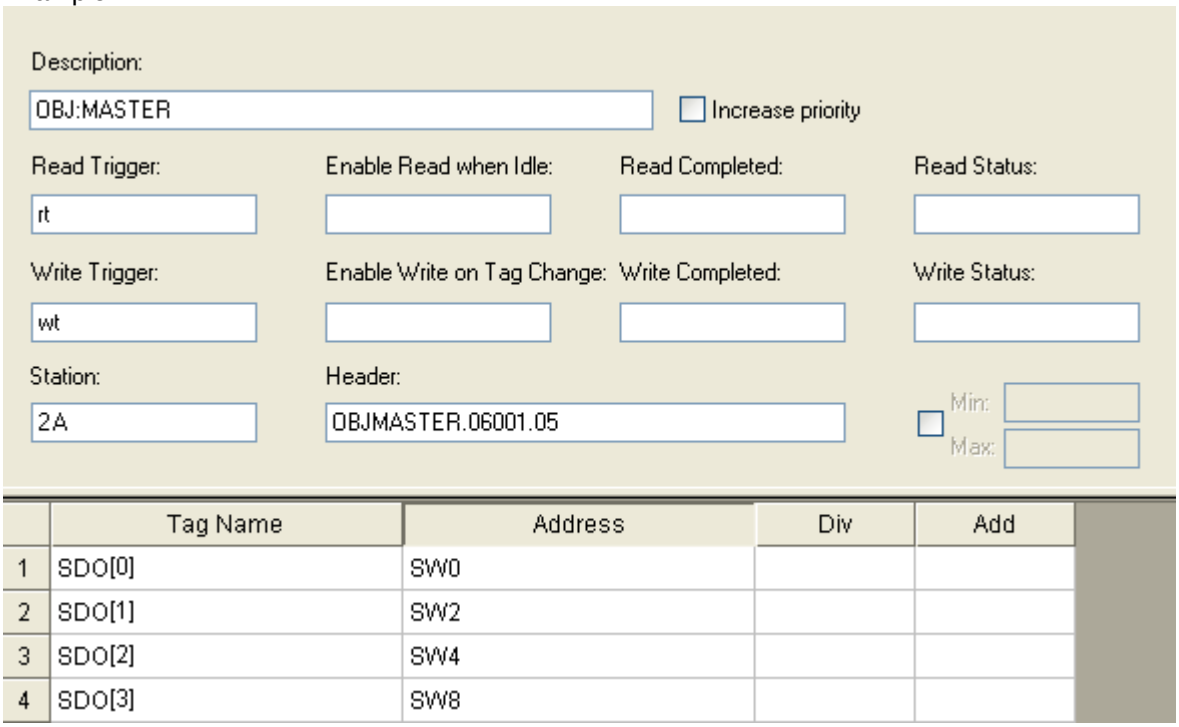
OBJMASTER.<Index>.<Sub-Index> : < Offset>

Where:
Index: Object Dictionary Index (HEX)

Sub- Index: Object Dictionary Sub-Index (HEX)
Offset: Address Offset

Address: For each table row (i.e., each tag/register association), configure the **Address** field with the can open message Data Byte. As Object Dictionary entries may contain data of arbitrary size, the address may range according to Register data size. All types configured for CANMASTER header are also valid for OBJMASTER header.

Example:



	Tag Name	Address	Div	Add
1	SDO[0]	SW0		
2	SDO[1]	SW2		
3	SDO[2]	SW4		
4	SDO[3]	SW8		

CANOpen Messages - SLAVE

The syntax below shows how configure a CANOpen an entry of a simulated Object Dictionary. Once configured, it may answer for Read or Write Commands. If you need to configure more than one entry for Slave, you can create another Standard Driver Sheet.
For this driver version, Slave answers only for upload and download SDO Protocols. SDO Block Upload, SDO Block Upload and PDOs are not supported.

Station Syntax

<Node ID>: Simulated device node ID (HEX)

I/O Address Syntax

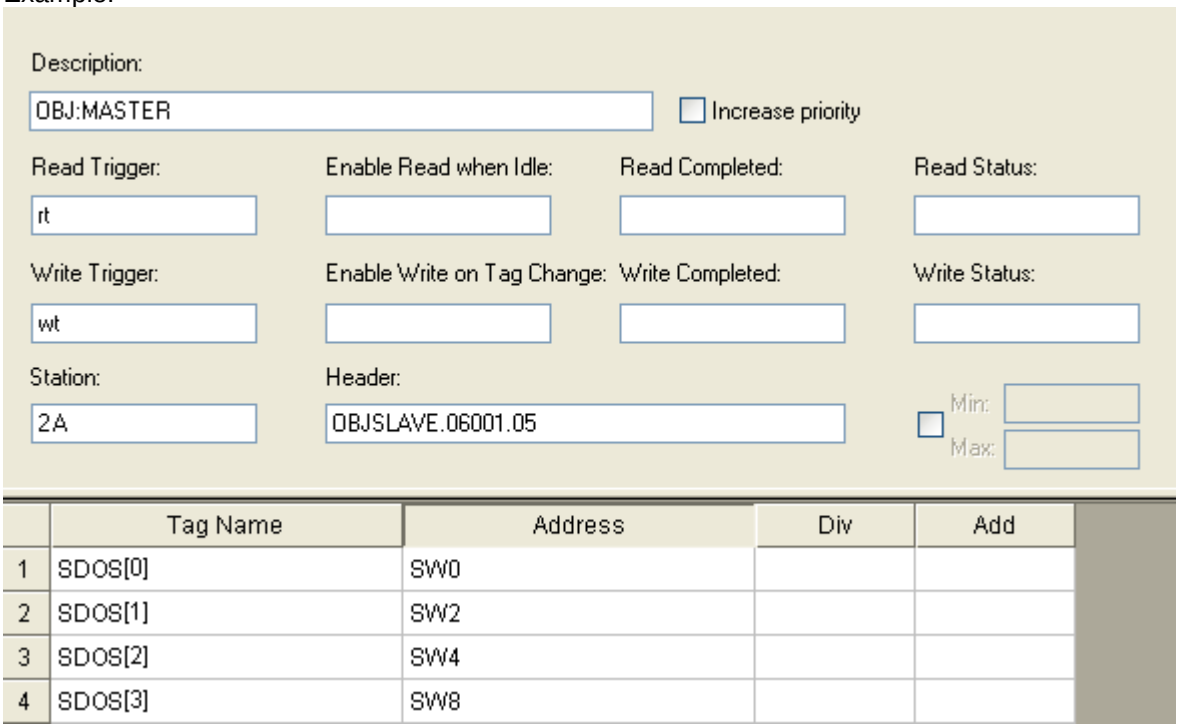
OBJMASTER.<Index>.<Sub-Index >

Where:
Index: Object Dictionary Index (HEX)

Sub- Index: Object Dictionary Sub-Index (HEX)

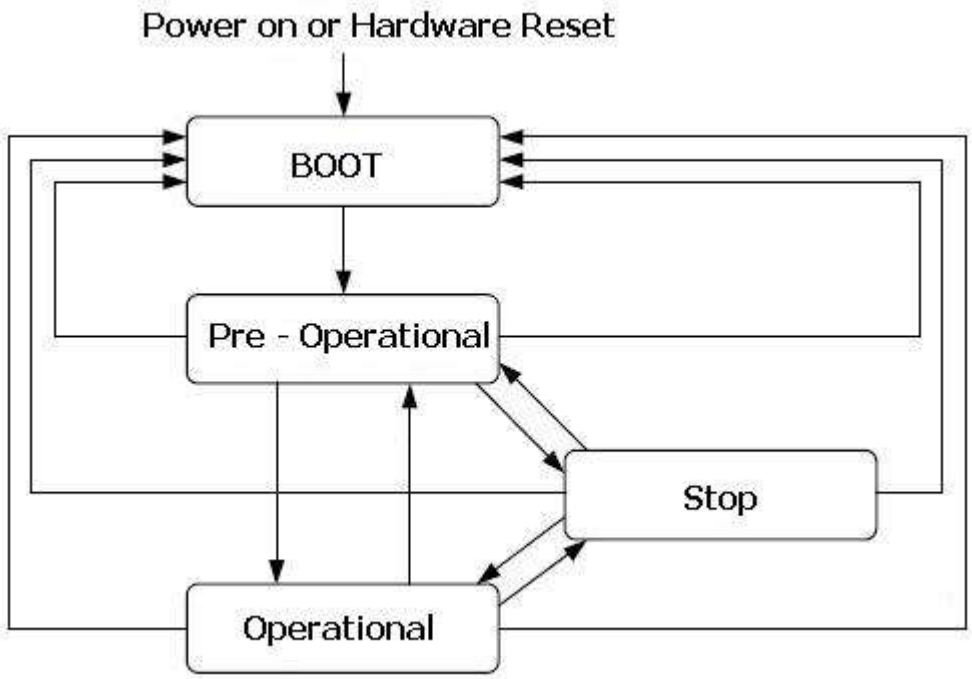
Address: For each table row (i.e., each tag/register association), configure the **Address** field with the can open message Data Byte. You can configure your OD node with an arbitrary data size, and also, you can use any of types configured for CANMaster header.

Example:



CANOpen NMT Message

NMT are control messages, which are used to get or to set a CANOpen device state. According to Cia 301 Specification, there CANOpen PLC has four states: BOOT, PRE-OPERATIONAL, OPERATIONAL and STOP



Header

NMT: < Offset>

Offset: Address Offset

Station

Station field should be kept empty.

Address: For each table row (i.e., each tag/register association), configure the **Address** field with the can open device node-id. You must use String type, and size should be greater or equal than 9

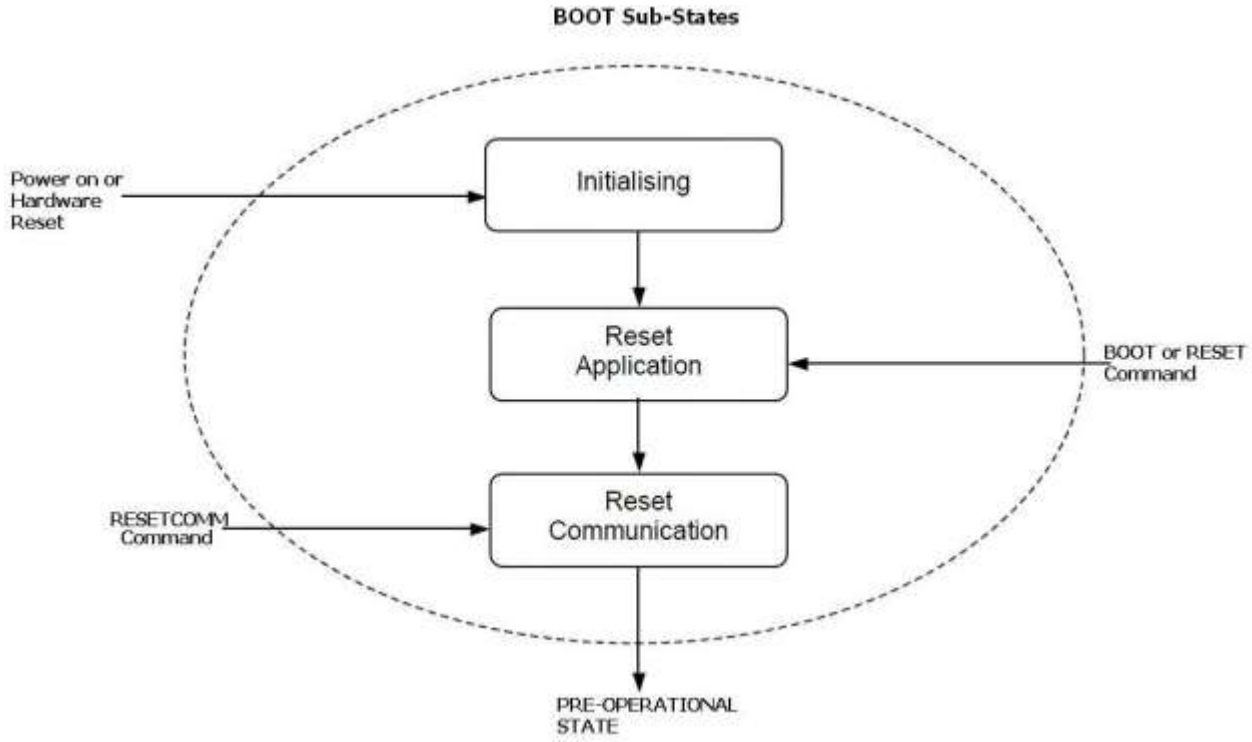
S<Node-ID>.9

Tags must be strings. If you read a device state tags will assume one of the four values: “OPER”, “PRE-OPER”, “BOOT”, “STOP”, as shown in table below. For writing, you should enter one of these four values. For state Boot and Operational, you can also enter values Reset or Start, respectively, as shown in table below.

Write Command	Read Value	Get / Set State

BOOT / RESET	BOOT	Boot
STOP	STOP	Stop
PREOP	PREOP	Pre operational
OPER/START	OPER	Operational
RESETCOMM	BOOT	Reset communication

The “initialization” or “boot” state is divided into three sub-states in order to enable a complete or partial reset of a node.

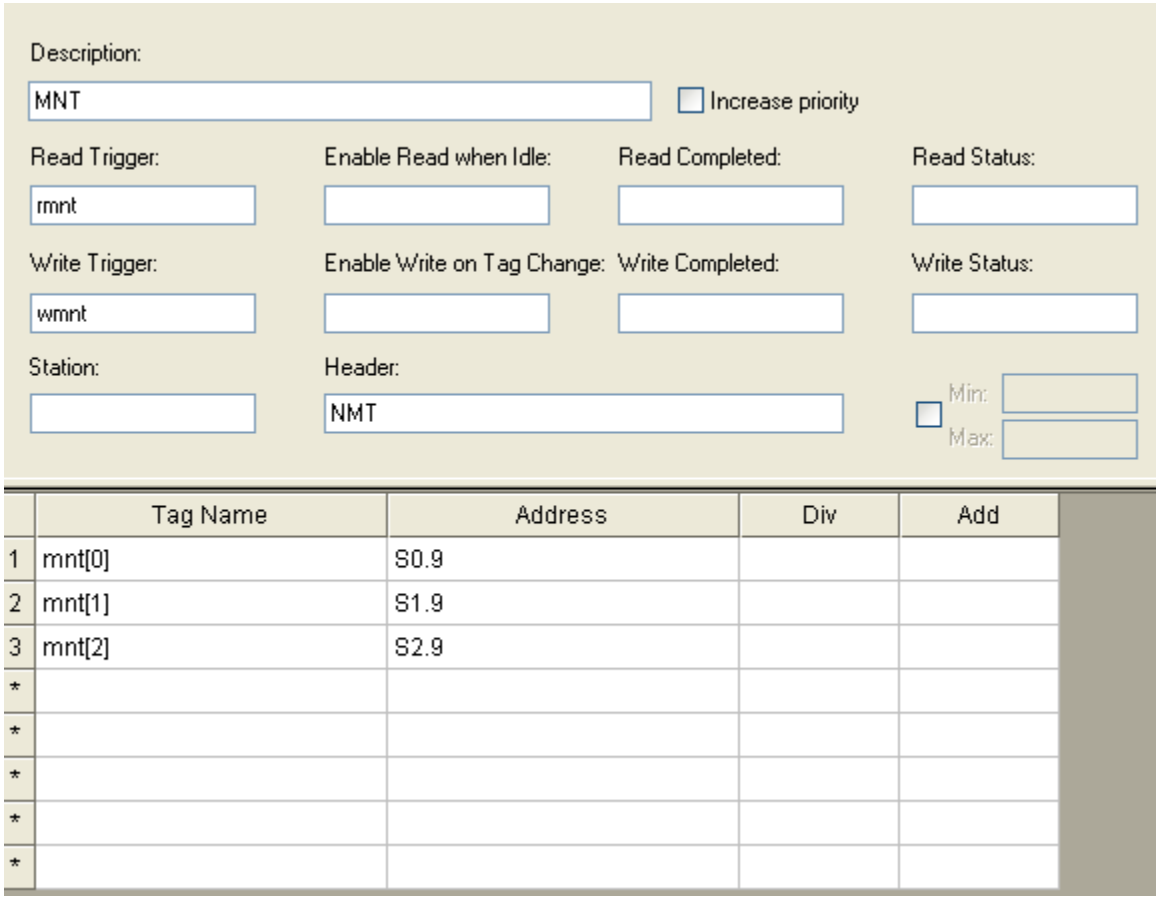


1. Initializing: This is the first sub-state the device enters after power-on or hardware reset. After finishing the basic node initialization the device enters autonomously into the state Reset_Application.

2. Reset_Application: In this state the parameters of the manufacturer specific profile area and of the standardized device profile area are set to their power-on values. After setting of the power-on values the state Reset_Communication is entered autonomously. Writing “RESET” or “BOOT”, device is enter in this sub-state

3. Reset_Communication: In this state the parameters of the communication profile area are set to their power-on values. After this the state Initialization is finished and the device executes the write boot-up object service and enters the state PRE-OPERATIONAL. Writing “RESETCOMM”, device enter in this substate

Example:

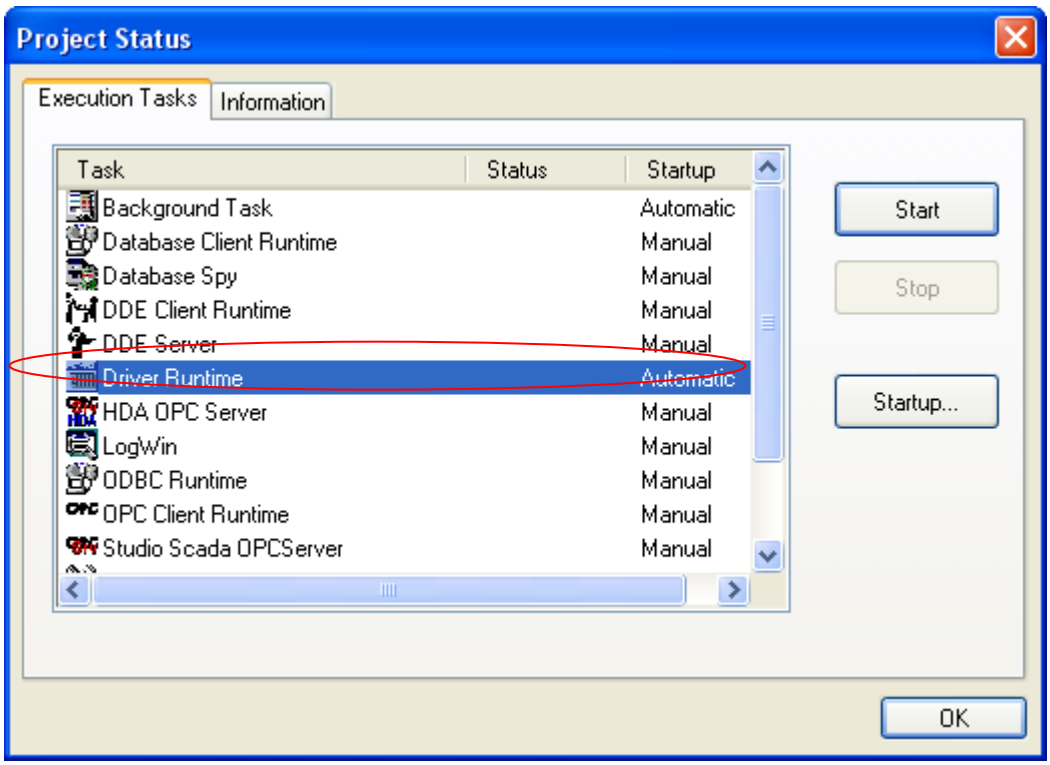

Description:
MNT
Increase priority
Read Trigger:
rmnt
Enable Read when Idle:
Read Completed:
Read Status:
Write Trigger:
wmnt
Enable Write on Tag Change:
Write Completed:
Write Status:
Station:
Header:
NMT
Min:
Max:
Tag Name
Address
Div
Add
1 mnt[0] S0.9
2 mnt[1] S1.9
3 mnt[2] S2.9

Executing the Driver

By default, Studio will automatically execute your selected communication driver(s) during application runtime. However, you may verify your application's runtime execution settings by checking the *Project Status* dialog.

To verify that the communication driver(s) will execute correctly:

1. From the main menu bar, select **Project** → **Status**. The *Project Status* dialog displays:



Project Status Dialog

2. Verify that the *Driver Runtime* task is set to **Automatic**.
   - If the setting is correct, then proceed to step 3 below.
   - If the **Driver Runtime** task is set to **Manual**, then select the task and click the **Startup** button to toggle the task's *Startup* mode to **Automatic**.
3. Click **OK** to close the *Project Status* dialog.
4. Start the application to run the driver.

Troubleshooting

If the CAN driver fails to communicate with the target device, then the database tag(s) that you configured for the **Read Status** or **Write Status** fields of the Main Driver Sheet will receive an error code. Use this error code and the following table to identify what kind of failure occurred.

Error Code	Description	Possible Causes	Procedure to Solve
0	OK	Communication without problems	Not applicable
1	General Error	Internal Driver error	Unexpected Error – please contact Technical Support
2	CAN ID not found	Message was not send by CAN device.	First of all, check if CAN ID is correctly configured in station field. If so, try to send a remote request (configure header as CANMASTER.REQNF).
3	Error Starting library	A library was not found or driver it is corrupted	Check if CANOpenAPI.dll is placed in DRV\Api folder. Check if device specific library (PCANBasic.dll for Peak and CIF50DLL.dll for Hilscher) is placed in DRV\API folder
4	Device Specific Error	Generic Error related to the PCANBasicAPI.dll (Peak) or CIF50DLL.DLL (Hilscher)	Check Protocol Analyzer to get Error Code and follow procedure described on the API error table
5	Invalid Parameter	There are two possible causes: 1) For CANMaster header, the options together REQRESET and ASYNC are invalid.; 2) An invalid command was passed to NMT header	For case 1, reconfigure CANMaster header. Choose other options like CANMASTE.REQRESET.SYNC For case 2, check if you tried to write a command different to RESET, BOOT, PREOP, OPER, START, or RESETCOMM for NMT header
6	Communication Error	Communication parameters in Settings are incorrectly configured	Check if Baud rate, Drv Interface and (if Peak) Peak communication parameters are correct.
7	Invalid Message Size	This error occurs for OBJMaster header. You are trying to read more bytes than arrived.	Reconfigure the sheet in order to read the correct number of bytes
8	Invalid Command	This error occurs if you try to read or write in Slave Headers (OBJSlave, CANSlave).	You should not read or write in slave headers.
9	Invalid Unsolicited Header	A read/write request has arrived for an inexistent or invalid Slave device	Check if Slave Station is correct (Settings->Advance->Station) so the driver can answer correctly for Slave Read/Write commands. For OBJSlave, check if parameters Index and Subindex are correctly configured.
-37	Invalid Header	This usually happens when you configure a string Tag in the **Header** field and the value of this tag does not comply with the header syntax	Change the value of the tag in the **Station** field to one that complies with the **Header** syntax
-38	Invalid Station	Invalid station specified in Station field	Specify a valid station in the Driver Worksheet.
-39	Block Size Error	You are trying to send a CAN message	Change the size of buffer (Settings->Advanced->Rx

		with more than 8 bytes or you are trying to read/write more bytes than configured in Settings->Advanced->Rx Buffer (CANOpen)	Buffer). Reconfigure the sheets in order to CANMessage have at most 8 bytes.
-9	No memory	The buffer configured in Settings->Advanced is to large	Reduce the buffer size
-15	Timeout Start Message	▪ Disconnected cables ▪ Wrong station number	▪ Check the cable wiring. ▪ Check the station number.

⇨ **Tip:**

You can monitor communication status by establishing an event log in Studio's *Output* window (*LogWin* module). To establish a log for **Field Read Commands**, **Field Write Commands** and **Serial Communication,** right-click in the *Output* window and select the desired options from the pop-up menu.

You can also use the *Remote LogWin* module (**Tools** → **Remote LogWin**) to establish an event log on a remote unit that runs Windows CE.

The following errors are extracted from the PEAK API, Hilscher API, and Advantech documentation. Please contact PEAK, Hilscher and Advantech technical support for possible causes and procedures to solve.

Peak API Errors

Error Code	Description
0	No error
1	Transmit buffer in CAN controller is full
2	CAN controller was read too late
4	Bus error: an error counter reached the 'light' limit
8	Bus error: an error counter reached the 'heavy' limit
16	Bus error: the CAN controller is in bus-off state
32	Receive queue is empty
64	Receive queue was read too late
128	Transmit queue is full
256	Test of the CAN controller hardware registers failed (no hardware found)
512	Driver not loaded
1024	Hardware already in use by a Net
2048	A Client is already connected to the Net
5020	Hardware handle is invalid
6144	Net handle is invalid
7168	Client handle is invalid
8192	Resource (FIFO, Client, timeout) cannot be created
16384	Invalid parameter
32768	Invalid parameter value
65536	Unknown error

262144	Channel is not initialized

Hilscher API errors

Error Code	Description
-1	Board not initialized
-2	Error in internal init state
-3	Error in internal read state
-4	Command on this channel is active
-5	Unknown parameter in function occurred
-6	Version is incompatible with DLL
-7	Error during PCI set run mode
-8	Could not read PCI dual port memory length
-9	Error during PCI set run mode
-10	Dual port ram not accessible(board not found)
-11	Not ready (ready flag failed)
-12	Not running (running flag failed)
-13	Watchdog test failed
-14	Signals wrong OS version
-15	Error in dual port flags
-16	Send mailbox is full
-17	PutMessage timeout
-18	GetMessage timeout
-19	No message available
-20	RESET command timeout
-21	COM-flag not set
-22	IO data exchange failed
-23	IO data exchange timeout
-24	IO data mode unknown
-25	Function call failed
-26	DPM size differs from configuration
-27	State mode unknown
-28	Output port already in use
-30	Driver not opened (device driver not loaded)
-31	Can't connect with device
-32	Board not initialized (DevInitBoard not called)
-33	IOCTRL function failed
-34	Parameter Device Number invalid
-35	Parameter Info Area unknown
-36	Parameter Number invalid
-37	Parameter Mode invalid
-38	NULL pointer assignment

-39	Message buffer too short
-40	Parameter Size invalid
-42	Parameter Size with zero length
-43	Parameter Size too long
-44	Device address null pointer
-45	Pointer to buffer is a null pointer
-46	SendSize parameter too long
-47	ReceiveSize parameter too long
-48	Pointer to buffer is a null pointer
-49	Pointer to buffer is a null pointer
-50	Memory allocation error
-51	Read I/O timeout
-52	Write I/O timeout
-53	PCI transfer timeout
-54	Download timeout
-55	Database download failed
-56	Firmware download failed
-57	Clear database on the device failed
-60	Virtual memory not available
-61	Unmap virtual memory failed
-70	General error
-71	General DMA error
-74	I/O WatchDog failed
-75	Device WatchDog failed
-80	Driver unknown
-81	Device name invalid
-82	Device name unknown
-83	Device function not implemented
-100	File not opened
-101	File size zero
-102	Not enough memory to load file
-103	File read failed
-104	File type invalid
-105	File name not valid
-110	Firmware file not opened
-111	Firmware file size zero
-112	Not enough memory to load firmware file
-113	Firmware file read failed
-114	Firmware file type invalid
-115	Firmware file name not valid
-116	Firmware file download error

-117	Firmware file not found in the internal table
-118	Firmware file BOOTLOADER active
-119	Firmware file no file path
-120	Configuration file not opend
-121	Configuration file size zero
-122	Not enough memory to load configuration file
-123	Configuration file read failed
-124	Configuration file type invalid
-125	Configuration file name not valid
-126	Configuration file download error
-127	No flash segment in the configuration file
-128	Configuration file differs from database
-131	Database size zero
-132	Not enough memory to upload database
-133	Database read failed
-136	Database segment unknown
-150	Version of the descript table invalid
-151	Input offset is invalid
-152	Input size is 0
-153	Input size does not match configuration
-154	Invalid output offset
-155	Output size is 0
-156	Output size does not match configuration
-157	Station not configured
-158	Cannot get the Station configuration
-159	Module definition is missing
-160	Empty slot mismatch
-161	Input offset mismatch
-162	Output offset mismatch
-163	Data type mismatch
-164	Module definition is missing,(no Slot/Idx)

Advantech Error Codes

Error Code	Description
995	Users cancel the operation or reset chip while drivers are receiving data
31	Busoff of device is discovered before drivers read any frames
87	Drivers cannot allocate resources according to the number defined by the third parameter frame
997	In asynchronous mode, operation will be pending if drivers cannot complete user's write request at present

Revision History

Doc. Revision	Driver Version	Author	Date	Description of Changes
A	1.00	Fellipe Peternella	Aug. 03, 2010	Initial version
B	1.01	André Körbes	Sep. 16, 2010	Included header CANBATCH Fixed "TPC" interface for Advantech devices on WinCE.
C	1.2	Paulo Balbino	Jan. 26, 2015	Added support for the CANSLAVE mode to handle multiple messages with different IDs
D	1.3	Priya Yennam	Jan. 14, 2016	Fixed CAN driver performance issue of driver being incredibly slow.