

# Hilscher Universal Driver

© 2017 PTC Inc. All Rights Reserved.

# Table of Contents

<b>Hilscher Universal Driver</b> .....	<b>1</b>
<b>Table of Contents</b> .....	<b>2</b>
Hilscher Universal Driver .....	5
Overview .....	5
External Dependencies .....	5
<b>Channel Setup</b> .....	<b>6</b>
Channel Properties — General .....	6
Channel Properties — Write Optimizations .....	7
Channel Properties — Advanced .....	8
Channel Properties — Board Selection .....	8
Channel Properties — SyCon Database .....	9
I/O Data References .....	10
Addressing Type .....	11
Bit Reference Tags .....	12
8-Bit Data Expansion .....	14
16-Bit Data Expansion .....	16
32-Bit Data Expansion .....	17
<b>Device Setup</b> .....	<b>20</b>
Device Properties — General .....	20
Device Properties — Scan Mode .....	22
Device Properties — Tag Generation .....	22
Automatic Tag Database Generation .....	24
Device Properties — Timing .....	26
Device Properties — Auto-Demotion .....	27
Device Properties — Device Type .....	27
<b>Data Types Description</b> .....	<b>28</b>
<b>Address Descriptions</b> .....	<b>29</b>
Process Image Address Descriptions .....	29
IEC Address Descriptions .....	34
<b>Error Descriptions</b> .....	<b>39</b>
Error Codes .....	40
Missing address .....	43
Device address '<address>' contains a syntax error .....	43
Address '<address>' is out of range for the specified device or register .....	43
Data Type '<type>' is not valid for device address '<address>' .....	43
Device address '<address>' is Read Only .....	43

Array size is out of range for address '<address>' .....	44
Array Support is not available for the specified address: '<address>' .....	44
Unable to load '<dll>' .....	44
Unable to import from '<dll>' .....	44
DevOpenDriver () failed with error code '<code>' .....	45
Memory allocation error .....	45
Device '<device name>' is not responding .....	45
Unable to read device info data in area '<area>'. Board '<board>' returned Error Code '<code>' ..	46
Unable to read '<block size>' device info bytes in area '<area>'. Board '<board>' returned Error Code '<code>' .....	46
Unable to read task state data in task '<task num>'. Board '<board>' returned Error Code '<code>'	47
Unable to read '<block size>' task state bytes in task '<task num>'. Board '<board>' returned Error Code '<code>' .....	47
Unable to read tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' .....	47
Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' .....	48
Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' .....	48
Unable to read tag '<name>': msg.b=<command>, msg.device_adr=<Device ID>... .....	48
Unable to read '<block size>' message bytes: msg.b=<command>, msg.device_adr=<Device ID>... ..	49
Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'] .....	49
Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'] .....	49
Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'] .....	50
Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>'] .....	50
Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>'] .....	50
Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>'] .....	51
The file is not a valid Sycon database or may be corrupt .....	51
Auto tag database generation cannot be performed while the driver is processing tags .....	51
Board Type for Board '<board number>' does not match the actual board installed. Verify Board Type and/or Board Selection .....	52
Board Type for Board '<board number>' does not match the Slave Type for one or more Slaves configured. Delete or edit Slaves accordingly .....	52
dbm32.dll is not loaded and is required for auto tag generation. Verify SyCon is installed .....	52

**Appendix: Slave Board Configuration** ..... 53  
**Index** ..... 67

## Hilscher Universal Driver

---

Help version 1.028

### CONTENTS

#### Overview

What is the Hilscher Universal Driver?

#### Channel Setup

How do I configure a channel for use with this driver?

#### Device Setup

How do I configure a device for use with this driver?

#### Data Types Description

What data types does this driver support?

#### Address Descriptions

How do I address a data location from a master / slave device?

#### Automatic Tag Database Generation

How can I easily configure tags for the Hilscher Universal Driver?

#### Error Descriptions

What error messages does the Hilscher Universal Driver produce?

### Overview

---

The Hilscher Universal Driver provides a reliable way to connect Hilscher Universal devices to OPC client applications; including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with Hilscher Communications Interface (CIF) cards. I/O and diagnostic information is available through the OPC server. The CIF cards currently supported are DeviceNet Master / Slave and Profibus DP Master / Slave.

### External Dependencies

---

This driver has external dependencies. It requires that SyCon (Hilscher's System Configuration Software) and the correct PCI interface card (CIF 50 models) be installed on the same machine as the OPC server.

## Channel Setup

A channel represents the SyCon Configuration Database, which includes the board assignment and device/module definitions. The device/module definitions are imported from the configuration file. For more information, refer to [SyCon Database Import](#).

Select a link from the list below for information on a specific aspect of Channel Setup.

### [Board Selection](#)

How do I select the board number and bus type over which communications will occur?

### [Slave Board Configuration](#)

How do I configure a local Hilscher Slave board? How do I configure the server to communicate directly with the local Slave board?

### [SyCon Database Import](#)

How do I specify the location of the SyCon Configuration Database?

## Channel Properties — General

This server supports the use of simultaneous multiple communications drivers. Each protocol or driver used in a server project is called a channel. A server project may consist of many channels with the same communications driver or with unique communications drivers. A channel acts as the basic building block of an OPC link. This group is used to specify general channel properties, such as the identification attributes and operating mode.

Property Groups	<input type="checkbox"/> <b>Identification</b>	
<b>General</b>	Name	
Write Optimizations	Description	
Advanced	Driver	
	<input type="checkbox"/> <b>Diagnostics</b>	
	Diagnostics Capture	Disable

### Identification

**Name:** User-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information.

• For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.

**Description:** User-defined information about this channel.

• Many of these properties, including Description, have an associated system tag.

**Driver:** Selected protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties.

• **Note:** With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-

acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

## Diagnostics

**Diagnostics Capture:** When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.

● **Note:** This property is disabled if the driver does not support diagnostics.

● *For more information, refer to "Communication Diagnostics" in the server help.*

## Channel Properties — Write Optimizations

As with any OPC server, writing data to the device may be the application's most important aspect. The server intends to ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties that can be used to meet specific needs or improve application responsiveness.

Property Groups	[-] <b>Write Optimizations</b>	
General	Optimization Method	Write Only Latest Value for All Tags
<b>Write Optimizations</b>	Duty Cycle	10

## Write Optimizations

**Optimization Method:** controls how write data is passed to the underlying communications driver. The options are:

- **Write All Values for All Tags:** This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags:** Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.
  - **Note:** This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags:** This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

**Duty Cycle:** is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

● **Note:** It is recommended that the application be characterized for compatibility with the write optimization enhancements before being used in a production environment.

## Channel Properties — Advanced

This group is used to specify advanced channel properties. Not all drivers support all properties; so the Advanced group does not appear for those devices.

Property Groups	[-] <b>Non-Normalized Float Handling</b>	
General	Floating-Point Values	Replace with Zero
Write Optimizations	[-] <b>Inter-Device Delay</b>	
<b>Advanced</b>	Inter-Device Delay (ms)	0

**Non-Normalized Float Handling:** Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. A non-normalized value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. The default is Replace with Zero. Drivers that have native float handling may default to Unmodified. Descriptions of the options are as follows:

- **Replace with Zero:** This option allows a driver to replace non-normalized IEEE-754 floating point values with zero before being transferred to clients.
- **Unmodified:** This option allows a driver to transfer IEEE-754 denormalized, normalized, non-number, and infinity values to clients without any conversion or changes.

● **Note:** This property is disabled if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM data is not affected by this setting.

● *For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.*

**Inter-Device Delay:** Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

● **Note:** This property is not available for all drivers, models, and dependent settings.

## Channel Properties — Board Selection

Property Groups	[-] <b>Board Selection</b>	
General	Board	0
<b>Board Selection</b>	Type	DeviceNet Master
SyCon Database		



**Board:** Specify the board on which communications should occur for the given channel. In the drop-down menu, Board x correlates to Board x in SyCon.

**Type:** Specify the selected board's bus type and master/slave type (if applicable).

### Supported Board Types

DeviceNet Master	Local DeviceNet Master board (such as CIF50-DNM).	Read/Write to DeviceNet slave I/O.
DeviceNet Slave	Local DeviceNet Slave board (CIF50-DNS).	Read/Write to local board I/O only.
Profibus DP Master	Local Profibus Master board (such as CIF50-PB).	Read/Write to Profibus slave I/O.
Profibus DP Slave	Local Profibus Slave board (CIF50-DPS).	Read/Write to local board I/O only.

### Channel Properties — SyCon Database

Property Groups General Write Optimizations Advanced Board Selection <b>SyCon Database</b>	<div style="border: 1px solid gray; padding: 5px;"> <div style="border-bottom: 1px solid gray; margin-bottom: 5px;"> <span style="font-size: 1.2em;">▣</span> <b>Options</b> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Expanded SyCon Tag Import</td> <td style="text-align: right;"><b>Enable</b></td> </tr> <tr> <td>Addressing Type</td> <td style="text-align: right;">Process Image Offsets</td> </tr> <tr> <td>Bit Reference Tags</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>Word Reference Tags for 8-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>DWord Reference Tags for 8-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>Byte Reference Tags for 16-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>DWord Reference Tags for 16-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>Byte Reference Tags for 32-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> <tr> <td>Word Reference Tags for 32-bit I/O</td> <td style="text-align: right;">Disable</td> </tr> </table> <div style="border-top: 1px solid gray; margin-top: 5px;"> <span style="font-size: 1.2em;">▣</span> <b>SyCon Database</b> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Tag Import File</td> <td style="text-align: right;">Enter or Browse for SyCon Database File</td> </tr> <tr> <td>Synchronize</td> <td style="text-align: right;"><a href="#">Synchronize tags</a></td> </tr> </table> </div>	Expanded SyCon Tag Import	<b>Enable</b>	Addressing Type	Process Image Offsets	Bit Reference Tags	Disable	Word Reference Tags for 8-bit I/O	Disable	DWord Reference Tags for 8-bit I/O	Disable	Byte Reference Tags for 16-bit I/O	Disable	DWord Reference Tags for 16-bit I/O	Disable	Byte Reference Tags for 32-bit I/O	Disable	Word Reference Tags for 32-bit I/O	Disable	Tag Import File	Enter or Browse for SyCon Database File	Synchronize	<a href="#">Synchronize tags</a>
Expanded SyCon Tag Import	<b>Enable</b>																						
Addressing Type	Process Image Offsets																						
Bit Reference Tags	Disable																						
Word Reference Tags for 8-bit I/O	Disable																						
DWord Reference Tags for 8-bit I/O	Disable																						
Byte Reference Tags for 16-bit I/O	Disable																						
DWord Reference Tags for 16-bit I/O	Disable																						
Byte Reference Tags for 32-bit I/O	Disable																						
Word Reference Tags for 32-bit I/O	Disable																						
Tag Import File	Enter or Browse for SyCon Database File																						
Synchronize	<a href="#">Synchronize tags</a>																						

### Options

- [I/O Data References](#)
- [Addressing Type](#)
- [Bit Reference Tags](#)
- [8-Bit Data Expansion](#)
- [16-Bit Data Expansion](#)
- [32-Bit Data Expansion](#)

### SyCon Database

- **Tag Import File:** Specify the exact location of the SyCon configuration database from which the tags are imported. This file is used when Automatic Tag Database Generation is instructed to create the tag database.
  - **Note:** To configure I/O for a Hilscher Slave board (CIF50-DNS), a separate SyCon configuration database must be created for each Slave board. This is not to be confused with the SyCon

configuration database set up for the Master. When a channel's board type is specified as a Slave-type, the SyCon Database specified must correspond specifically to this Slave. For more information, refer to [Slave Board Configuration](#).

- **Synchronize:** clicking "Synchronize tags" will generate tags for the channel.

## Supported Databases

Profibus-DP Database Extension: .pb

DeviceNet Database Extension: .dn

• See Also: [Automatic Tag Database Generation](#)

## I/O Data References

### Expanded SyCon Tag Import

When disabled, this option only imports the I/O symbolic names that are configured in SyCon. Tags are generated under the following folder:

*IO\<bus dependent module path>\*

When enabled, this option imports the I/O Symbolic Names configured in SyCon and also generates alternative references for each symbolic name. These alternative references (or Expansion Tags) provide different ways of looking at the same piece of data. Tags are generated under the following folders:

*IO\<bus dependent module path>\PI*

*IO\<bus dependent module path>\IEC*

• **Note:** Depending on the Addressing Type, tags may be generated to both of these folders. For more information, refer to [Addressing Type](#).

## Examples

### SyCon

Byte Addressing should be assumed. For more information on Byte and Word addressing modes, refer to [Address Descriptions](#). Details are as follows:

- **Configured I/O Module:** Word Array, QW, Length 4, Offset 0.
- **Symbolic Names:** Default is Output001, Offset 0, Word.

### OPC Server

For Expansion Tags, 16 bit Module Data / Byte References should be assumed. The following tags are generated in the OPC server under the specific options:

- **Import only SyCon I/O Tags:** Output001, Offset 0, Word.
- **Expanded SyCon Tag Import:** Output001, Offset 0, Word; Output001\_B0\_QB, Offset 0, Byte; Output001\_B1\_QB, Offset 1, Byte.

## Symbolic Names and Expansion Tags Differences

SyCon creates default symbolic names based on the Configured I/O. Users can create additional symbolic names (such as Bit, Word, DWord and String tags) within SyCon. Each SyCon symbolic name is imported as an individual tag in the OPC server. Expansion tags are an expansion of the symbolic name tags imported, regardless of whether or not the symbolic name tag is an expansion of the Configured I/O.

## Examples

### SyCon

Byte addressing should be assumed.

- **Configured I/O Module:** Byte Array, IB, Length 10, Offset 0.
- **Symbolic Names:** Default: Input001, Offset 0, Byte.
  - **Note:** These were created for the module MyWordTag, Offset 0, Word.

### OPC Server

For Expansion Tags, Generate Bit References should be assumed. The following tags are generated in the OPC server under the specific options:

- **Import only SyCon I/O Tags:** Input001, Offset 0, Byte; MyWordTag, Offset 0, Word.
- **Import and Expand SyCon I/O Tags:**
  - Input001, Offset 0, Byte
  - Input001\_IX\_00, Offset 0, Bit 0
  - Input001\_IX\_01, Offset 0, Bit 1
  - ...
  - Input001\_IX\_07, Offset 0, Bit 7
  - MyWordTag, Offset 0, Word
  - MyWordTag\_B0\_IX\_00, Offset 0, Bit 0
  - ...
  - MyWordTag\_B0\_IX\_07, Offset 0, Bit 7
  - MyWordTag\_B1\_IX\_00, Offset 1, Bit 0
  - ...
  - MyWordTag\_B1\_IX\_07, Offset 0, Bit 7

• **Note:** The example above shows how all symbolic name tags are expanded based on the chosen properties. Only bit references were generated additionally.

• **Important:** Using Expansion Tags should eliminate the need to create additional symbolic names in SyCon. The exception to this is Strings, since these are not included in the Expansion Settings.

## Addressing Type

There are two types of I/O addresses supported in the Hilscher Universal Driver: Process Image Offset Addressing and IEC Addressing. Descriptions are as follows:

- **Process Image Offset Addressing:** Address mnemonic and offsets are based on the physical offset into the Master's Process Image Memory Map (I/O Data). Addresses are always byte-based, regardless of the addressing mode selected in SyCon's Master Settings.

*IO\<bus dependent module path>\PI*

• **Note:** Expansion Tags are generated in a tag group labeled "PI". All tag addresses in this group are based on Process Image Addressing.

- **IEC Addressing:** Address mnemonic and offsets are based on standard Siemens addressing (IB, IW, ID). Addresses are byte-based or word-based, depending on the addressing mode selected in SyCon's Master Settings.

*IO\<bus dependent module path>\IEC*

• **Note:** Expansion Tags are generated in a tag group labeled "IEC". All tag addresses in this group are based on IEC Addressing.

**Important:** Expansion tags are generated for both addressing types. Two tag groups, "PI" and "IEC," will exist along with their respective Expansion Tags.

**Note:** For more information, refer to [Address Descriptions](#).

## Bit Reference Tags

A module's Byte, Word and DWord I/O data can have individual bits referenced. This option will automatically generate bit references at the same offsets as the Byte, Word and DWord reference. These tags will have the mnemonic IOX/OOX for Process Image Offset and IX/QX for IEC Offsets.

The examples below display how Bit tags are generated for Byte, Word and DWord I/O data. Input001, Input001AsWord and Input001AsDWord are symbolic names defined in the SyCon database. Module1 is a Byte Module.

### Example One: Bits Generated for Input001 (IOB11)

Tag Name	Address	Data Type
Input001_IOB	IOB0011	Char
Input001_IOX_00	IOX0011.00	Boolean
Input001_IOX_01	IOX0011.01	Boolean
Input001_IOX_02	IOX0011.02	Boolean
Input001_IOX_03	IOX0011.03	Boolean
Input001_IOX_04	IOX0011.04	Boolean
Input001_IOX_05	IOX0011.05	Boolean
Input001_IOX_06	IOX0011.06	Boolean
Input001_IOX_07	IOX0011.07	Boolean
Input001AsDWord IOD	IOD0011	Boolean

**Note:** Bits 0-7 of Byte 11 are referenced individually in IOX tags.

### Example Two: Bits Generated for Input001AsWord (IOW12)

Tag Name	Address	Data Type
Input001AsDWord_B0_IOX_00	IOX0011.00	Boolean
Input001AsDWord_B0_IOX_01	IOX0011.02	Boolean
Input001AsDWord_B0_IOX_02	IOX0011.03	Boolean
Input001AsDWord_B0_IOX_03	IOX0011.04	Boolean
Input001AsDWord_B0_IOX_04	IOX0011.04	Boolean
Input001AsDWord_B0_IOX_05	IOX0011.05	Boolean
Input001AsDWord_B0_IOX_06	IOX0011.06	Boolean
Input001AsDWord_B0_IOX_07	IOX0011.07	Boolean
Input001AsDWord_B1_IOX_00	IOX0012.00	Boolean
Input001AsDWord_B1_IOX_01	IOX0012.01	Boolean
Input001AsDWord_B1_IOX_02	IOX0012.02	Boolean
Input001AsDWord_B1_IOX_03	IOX0012.03	Boolean
Input001AsDWord_B1_IOX_04	IOX0012.04	Boolean
Input001AsDWord_B1_IOX_05	IOX0012.05	Boolean
Input001AsDWord_B1_IOX_06	IOX0012.06	Boolean
Input001AsDWord_B1_IOX_07	IOX0012.07	Boolean
Input001AsDWord_IOW	IOW0011	Word
Input002_IOB	IOB0012	Char

● **Note:** Bits 0-15 of Word 11 are referenced individually in IOX tags. Since 'Module 1' is a Byte Module, Word 12 must be broken up into its individual Bytes (which are further referenced as Bits 0-7). If 'Module 1' were a Word Module, Bits 0-15 could be referenced.

**Example Three: Bits Generated for Input001AsDWord (IOD11)**

Tag Name	Address	Data Type
Input001AsDWord_IOD	IOD0011	DWord
Input001AsDWord_W0B0_IOX_00	IOX0011.00	Boolean
Input001AsDWord_W0B0_IOX_01	IOX0011.01	Boolean
Input001AsDWord_W0B0_IOX_02	IOX0011.02	Boolean
Input001AsDWord_W0B0_IOX_03	IOX0011.03	Boolean
Input001AsDWord_W0B0_IOX_04	IOX0011.04	Boolean
Input001AsDWord_W0B0_IOX_05	IOX0011.05	Boolean
Input001AsDWord_W0B0_IOX_06	IOX0011.06	Boolean
Input001AsDWord_W0B0_IOX_07	IOX0011.07	Boolean
Input001AsDWord_W0B1_IOX_00	IOX0012.00	Boolean
Input001AsDWord_W0B1_IOX_01	IOX0012.01	Boolean
Input001AsDWord_W0B1_IOX_02	IOX0012.02	Boolean
Input001AsDWord_W0B1_IOX_03	IOX0012.03	Boolean
Input001AsDWord_W0B1_IOX_04	IOX0012.04	Boolean
Input001AsDWord_W0B1_IOX_05	IOX0012.05	Boolean
Input001AsDWord_W0B1_IOX_06	IOX0012.06	Boolean
Input001AsDWord_W0B1_IOX_07	IOX0012.07	Boolean
Input001AsDWord_W1B0_IOX_00	IOX0013.00	Boolean
Input001AsDWord_W1B0_IOX_01	IOX0013.01	Boolean
Input001AsDWord_W1B0_IOX_02	IOX0013.02	Boolean
Input001AsDWord_W1B0_IOX_03	IOX0013.03	Boolean
Input001AsDWord_W1B0_IOX_04	IOX0013.04	Boolean
Input001AsDWord_W1B0_IOX_05	IOX0013.05	Boolean
Input001AsDWord_W1B0_IOX_06	IOX0013.06	Boolean
Input001AsDWord_W1B0_IOX_07	IOX0013.07	Boolean
Input001AsDWord_W1B1_IOX_00	IOX0014.00	Boolean
Input001AsDWord_W1B1_IOX_01	IOX0014.01	Boolean
Input001AsDWord_W1B1_IOX_02	IOX0014.02	Boolean
Input001AsDWord_W1B1_IOX_03	IOX0014.03	Boolean
Input001AsDWord_W1B1_IOX_04	IOX0014.04	Boolean
Input001AsDWord_W1B1_IOX_05	IOX0014.05	Boolean
Input001AsDWord_W1B1_IOX_06	IOX0014.06	Boolean
Input001AsDWord_W1B1_IOX_07	IOX0014.07	Boolean

**Note:** Bits 0-31 of DWord 11 are referenced individually in IOX tags. Because 'Module 1' is a Byte Module, DWord 12 must be broken up into its individual Bytes which are further referenced as Bits 0-7. If 'Module 1' were a DWord Module, Bits 0-31 could be referenced.

## 8-Bit Data Expansion

Byte references are automatically generated for 8 Bit I/O Data when the "Expanded SyCon Tag Import" option is set to enable under SyCon Database. Additionally, 8 Bit I/O Data can be referenced as 16 bit and 32 bit entities. These options will automatically generate Word and/or DWord references at the same offsets as the Byte reference.

## Word Reference Tags for 8-bit I/O

These tags will have the mnemonic IOW/OOW for Process Image Offset and IW/QW for IEC Offsets. In order to access the Byte I/O data as a Word, these tags must be generated.

### DWord Reference Tags for 8-bit I/O

These tags will have the mnemonic IOD/OOD for Process Image Offset and ID/QD for IEC Offsets. In order to access the Byte I/O data as a DWord, these tags must be generated.

Word and DWord references are only generated if the size of module in question (including the offset into the module) is big enough to reference as a Word or DWord. The module must be at least 2 bytes in size to reference the module's base offset as a Word and 4 bytes in size to reference the module's base offset as a DWord. If a module is 2 bytes in size, no DWord references are generated. Likewise, if the module is 1 byte in size, no Word references are generated.

### Examples

#### SyCon

Byte Addressing is assumed. For more information on Byte and Word Addressing Modes, refer to [Address Descriptions](#).

**Configured I/O Module:** Byte Array, IB, Length 6, Offset 0.

#### OPC Server

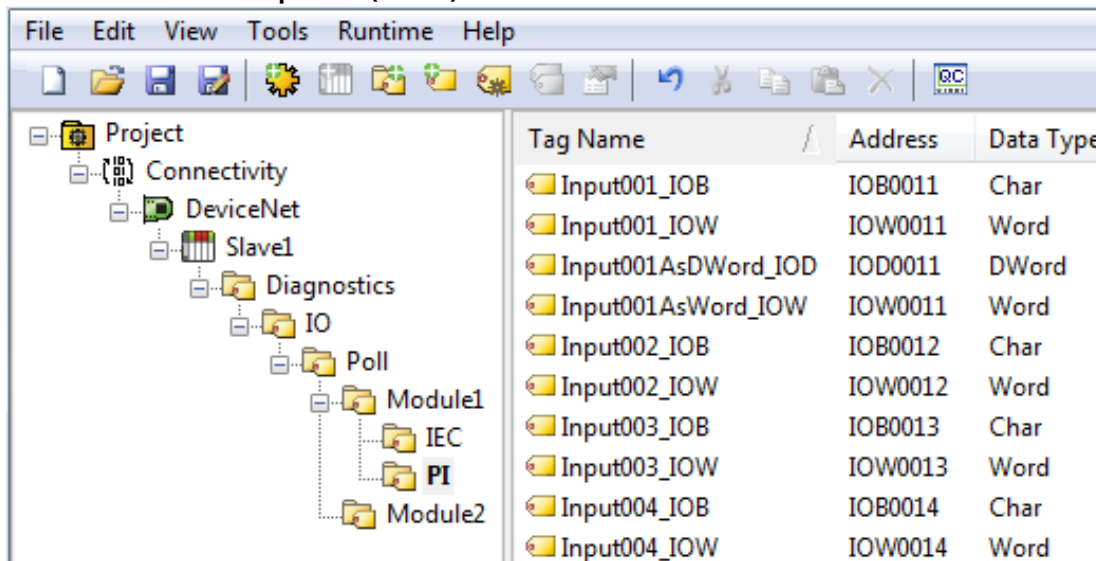
The following tags are generated in the OPC server.

Tag	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Byte Tags	IOB0	IOB1	IOB2	IOB3	IOB4	IOB5
Word Tags	IOW0	IOW1	IOW2	IOW3	IOW4	*
DWord Tags	IOD0	IOD1	IOD2	*	*	*

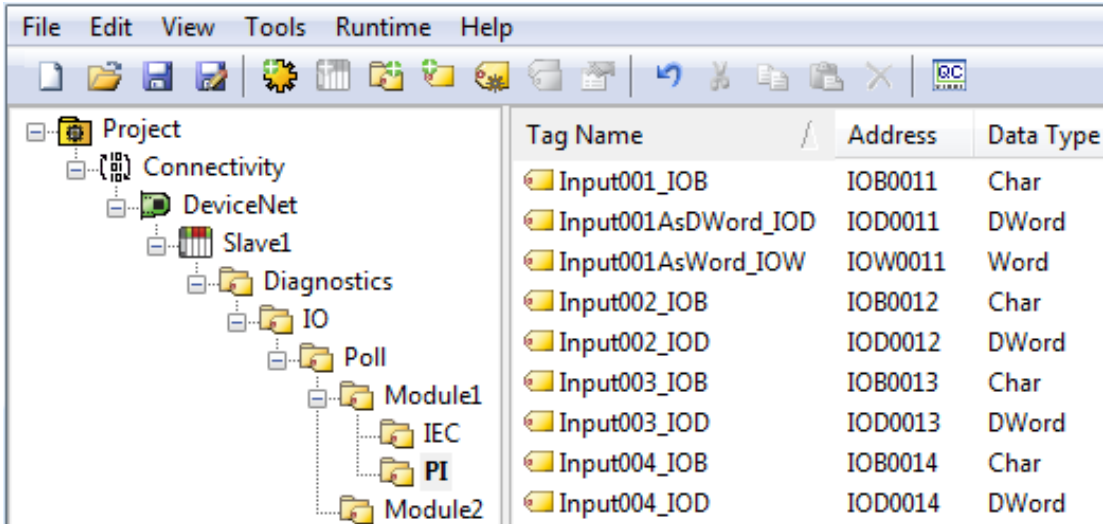
\*No tag is generated because it would exceed the size of the module.

The images below illustrate how Word and DWord tags are generated for Byte data. Input001 is a symbolic name defined in the SyCon database. When expanded, a Byte reference (Input001\_IOB) is automatically generated for Input001.

#### Word Generated for Input001 (IOB12)



#### DWord Generated for Input001 (IOB12)



## 16-Bit Data Expansion

Word references are automatically generated for 16 bit I/O Data if the "Expanded SyCon Tag Import" option is set to enable under SyCon Database. Additionally, 16 bit I/O Data can be referenced as 8 Bit and 32 bit entities. These options will automatically generate Byte and/or DWord references at the same offsets as the Word reference.

### Byte Reference Tags for 16-bit I/O

These tags will have the mnemonic IOB/OOB for Process Image Offset and IB/QB for IEC Offsets. In order to access the Bytes of Word I/O data, these tags must be generated.

### DWord Reference Tags for 16-bit I/O

These tags will have the mnemonic IOD/OOD for Process Image Offset and ID/QD for IEC Offsets. In order to access the Word I/O data as a DWord, these tags must be generated.

Byte and DWord references are only generated if the size of module in question (including the offset into the module) is big enough to reference as a DWord. The module must be at least 2 words in size to reference the module's base offset as a DWord. If a module is 1 word in size, no DWord references are generated.

## Examples

### SyCon

Byte addressing is assumed.

**Configured I/O Module:** Word Array, IW, Length 6, Offset 0.

### OPC Server

The following tags are generated in the OPC server.

Tags	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Byte Tags	IOB0	IOB1	IOB2	IOB3	IOB4	IOB5
Word Tags	IOW0	N/A*	IOW2	N/A*	IOW4	N/A*
DWord Tags	IOD0	N/A*	IOD2	N/A*	**	N/A*

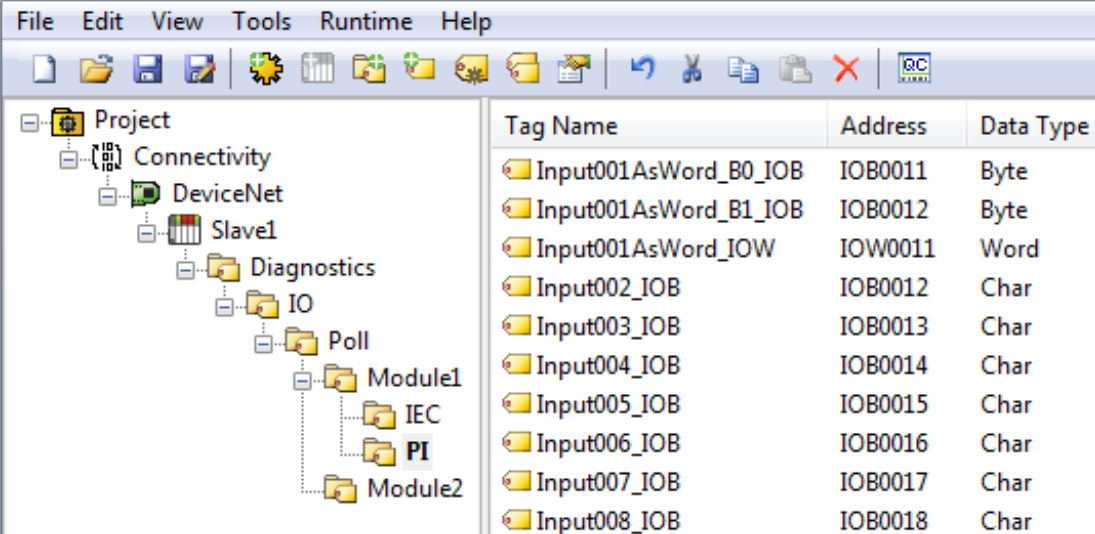


\*No tag is generated regardless of the size of the module. This follows the definition of a Word Module. For more information on Byte/Word/DWord Module Addressing, refer to [Address Descriptions](#).

\*\*No tag is generated because it would exceed the size of the module.

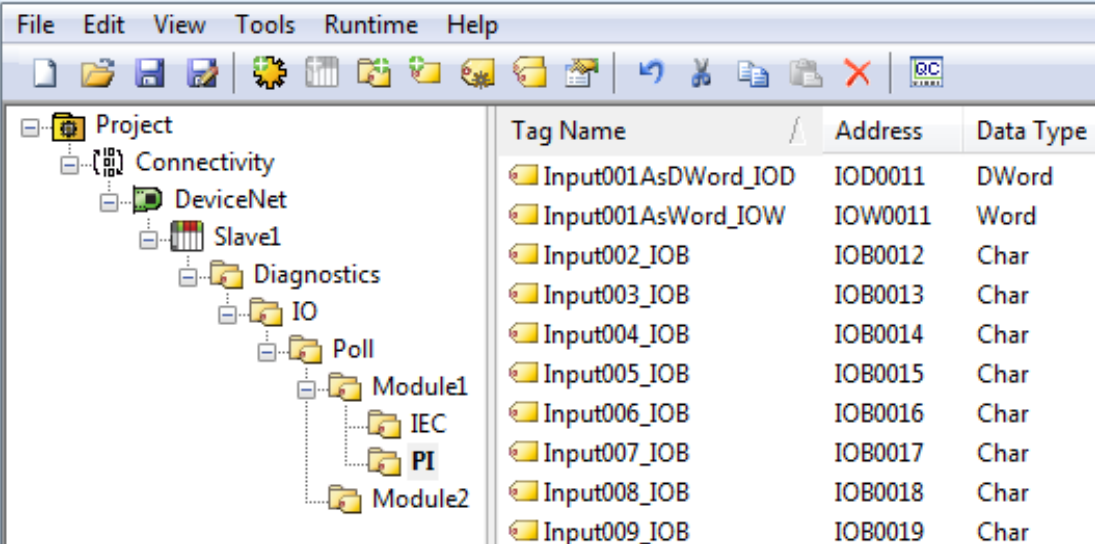
The images below illustrate how Byte and DWord tags are generated for Word data. Input001AsWord is a symbolic name defined in the SyCon database. When expanded, a Word reference (Input001AsWord\_IOW) is automatically generated for Input001AsWord.

#### Bytes Generated for Input001AsWord (IOW12)



Tag Name	Address	Data Type
Input001AsWord_B0_IOB	IOB0011	Byte
Input001AsWord_B1_IOB	IOB0012	Byte
Input001AsWord_IOW	IOW0011	Word
Input002_IOB	IOB0012	Char
Input003_IOB	IOB0013	Char
Input004_IOB	IOB0014	Char
Input005_IOB	IOB0015	Char
Input006_IOB	IOB0016	Char
Input007_IOB	IOB0017	Char
Input008_IOB	IOB0018	Char

#### DWord Generated for Input001AsWord (IOW12)



Tag Name	Address	Data Type
Input001AsDWord_IOD	IOD0011	DWord
Input001AsWord_IOW	IOW0011	Word
Input002_IOB	IOB0012	Char
Input003_IOB	IOB0013	Char
Input004_IOB	IOB0014	Char
Input005_IOB	IOB0015	Char
Input006_IOB	IOB0016	Char
Input007_IOB	IOB0017	Char
Input008_IOB	IOB0018	Char
Input009_IOB	IOB0019	Char

## 32-Bit Data Expansion

DWord references are automatically generated for 32 bit I/O Data if the "Expanded SyCon Tag Import" option is enabled under SyCon Database. Additionally, 32 bit I/O Data can be referenced as 8-Bit and 16 bit entities. These options will automatically generate Byte and/or Word references at the same offsets as the DWord reference.

### Byte Reference Tags for 32-bit I/O

These tags will have the mnemonic IOB/OOB for Process Image Offset and IB/QB for IEC Offsets. In order to access the Bytes of DWord Module data, these tags must be generated.

### Word Reference Tags for 32-bit I/O

These tags will have the mnemonic IOW/OOW for Process Image Offset and IW/QW for IEC Offsets. In order to access the Words of DWord Module data, these tags must be generated.

### Examples

#### SyCon

Byte Addressing is assumed.

**Configured I/O Module:** DWORD Array, ID, Length 4, Offset 0

#### OPC Server

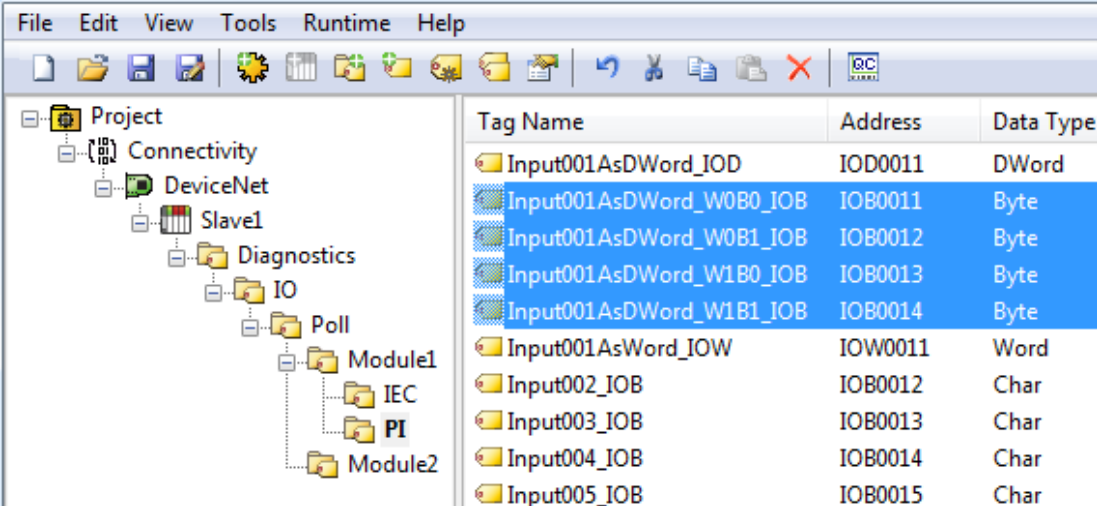
The following tags are generated in the OPC server.

Tags	Byte 0	Byte 1	Byte 2	Byte 3
Byte Tags	IOB0	IOB1	IOB2	IOB3
Word Tags	IOW0	N/A*	IOW2	N/A*
DWord Tags	IOD0	N/A*	N/A*	N/A*

\*No tag is generated regardless of the size of the module. This follows the definition of a Word Module. For more information on Byte/Word/DWord Module Addressing, refer to [Address Descriptions](#).

The examples below illustrate how Byte and Word tags are generated for DWord data. Input001AsDWord is a symbolic name defined in the SyCon database. When expanded, a Byte reference (Input001AsDWord\_IOD) is automatically generated for Input001AsDWord.

#### Bytes Generated for Input001AsDWord (IOD12)



Tag Name	Address	Data Type
Input001AsDWord_IOD	IOD0011	DWord
Input001AsDWord_W0B0_IOB	IOB0011	Byte
Input001AsDWord_W0B1_IOB	IOB0012	Byte
Input001AsDWord_W1B0_IOB	IOB0013	Byte
Input001AsDWord_W1B1_IOB	IOB0014	Byte
Input001AsWord_IOW	IOW0011	Word
Input002_IOB	IOB0012	Char
Input003_IOB	IOB0013	Char
Input004_IOB	IOB0014	Char
Input005_IOB	IOB0015	Char

#### Words Generated for Input001AsDWord (IOD12)

The screenshot shows a software interface with a menu bar (File, Edit, View, Tools, Runtime, Help) and a toolbar. On the left is a project tree with the following structure:

- Project
  - Connectivity
    - DeviceNet
      - Slave1
        - Diagnostics
          - IO
            - Poll
              - Module1
                - IEC
                - PI
              - Module2

On the right is a table with the following data:

Tag Name	Address	Data Type
Input001AsDWord_IOD	IOD0011	DWord
Input001AsDWord_W0_IOW	IOW0011	Word
Input001AsDWord_W1_IOW	IOW0013	Word
Input001AsWord_IOW	IOW0011	Word
Input002_IOB	IOB0012	Char
Input003_IOB	IOB0013	Char
Input004_IOB	IOB0014	Char
Input005_IOB	IOB0015	Char
Input006_IOB	IOB0016	Char
Input007_IOB	IOB0017	Char

## Device Setup

The device represents a single device in the SyCon Configuration Database. It can be a Master or a Slave.

### Connection Timeout

Specify the time that the driver will wait for a connection to be made with a device. Depending on network load, the connect time may vary with each connection attempt. The default setting is 3 seconds. The valid range is 1 to 30 seconds.

### Request Timeout

Specify the time that the driver will wait for a response from the device before giving up and going on to the next request. Longer timeouts only affect performance if a device is not responding. The default setting is 1000 milliseconds. The valid range is 100 to 30000 milliseconds.

### Retry Attempts

Specify the number of times that the driver will retry a message before giving up and going on to the next message. The default setting is 3 retries. The valid range is 1 to 10.

### Device IDs

The Device ID represents the MAC ID in SyCon. Its range varies from bus to bus. The Device ID allows Automatic Tag Database Generation to import the proper tags for a given device.

## Device Properties — General

A device represents a single target on a communications channel. If the driver supports multiple controllers, users must enter a device ID for each controller.

Property Groups	Identification	
General	Name	
Scan Mode	Description	
Auto-Demotion	Channel Assignment	
Redundancy	Driver	
	Model	
	ID Format	Decimal
	ID	2
	Operating Mode	
	Data Collection	Enable
	Simulated	No

### Identification

**Name:** This property specifies the name of the device. It is a logical user-defined name that can be up to 256 characters long, and may be used on multiple channels.

**Note:** Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".

**For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.**

**Description:** User-defined information about this device.

● Many of these properties, including Description, have an associated system tag.

**Channel Assignment:** User-defined name of the channel to which this device currently belongs.

**Driver:** Selected protocol driver for this device. This property specifies the driver selected during channel creation. It is disabled in the channel properties.

**Model:** This property specifies the specific type of device that is associated with this ID. The contents of the drop-down menu depends on the type of communications driver being used. Models that are not supported by a driver are disabled. If the communications driver supports multiple device models, the model selection can only be changed when there are no client applications connected to the device.

● **Note:** If the communication driver supports multiple models, users should try to match the model selection to the physical device. If the device is not represented in the drop-down menu, select a model that conforms closest to the target device. Some drivers support a model selection called "Open," which allows users to communicate without knowing the specific details of the target device. For more information, refer to the driver help documentation.

**ID:** This property specifies the device's station / node / identity / address. The type of ID entered depends on the communications driver being used. For many drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The ID format can be Decimal, Octal, and Hexadecimal. If the driver is Ethernet-based or supports an unconventional station or node name, the device's TCP/IP address may be used as the device ID. TCP/IP addresses consist of four values that are separated by periods, with each value in the range of 0 to 255. Some device IDs are string based. There may be additional properties to configure within the ID field, depending on the driver.

## Operating Mode

**Data Collection:** This property controls the device's active state. Although device communications are enabled by default, this property can be used to disable a physical device. Communications are not attempted when a device is disabled. From a client standpoint, the data is marked as invalid and write operations are not accepted. This property can be changed at any time through this property or the device system tags.

**Simulated:** This option places the device into Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The item's memory map is based on the group Update Rate. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

● **Notes:**

1. This System tag (\_Simulated) is read only and cannot be written to for runtime protection. The System tag allows this property to be monitored from the client.
2. In Simulation mode, the item's memory map is based on client update rate(s) (Group Update Rate for OPC clients or Scan Rate for native and DDE interfaces). This means that two clients that reference the same item with different update rates return different data.

🔴 Simulation Mode is for test and simulation purposes only. It should never be used in a production environment.

## Device Properties — Scan Mode

The Scan Mode specifies the subscribed-client requested scan rate for tags that require device communications. Synchronous and asynchronous device reads and writes are processed as soon as possible; unaffected by the Scan Mode properties.

Property Groups	☐ <b>Scan Mode</b>	
General	Scan Mode	Respect Client-Specified Scan Rate ▼
<b>Scan Mode</b>	Initial Updates from Cache	Disable

**Scan Mode:** specifies how tags in the device are scanned for updates sent to subscribed clients.

Descriptions of the options are:

- **Respect Client-Specified Scan Rate:** This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate:** This mode specifies the maximum scan rate to be used. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  - **Note:** When the server has an active client and items for the device and the scan rate value is increased, the changes take effect immediately. When the scan rate value is decreased, the changes do not take effect until all client applications have been disconnected.
- **Request All Data at Scan Rate:** This mode forces tags to be scanned at the specified rate for subscribed clients. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
- **Do Not Scan, Demand Poll Only:** This mode does not periodically poll tags that belong to the device nor perform a read to get an item's initial value once it becomes active. It is the client's responsibility to poll for updates, either by writing to the `_DemandPoll` tag or by issuing explicit device reads for individual items. *For more information, refer to "Device Demand Poll" in server help.*
- **Respect Tag-Specified Scan Rate:** This mode forces static tags to be scanned at the rate specified in their static configuration tag properties. Dynamic tags are scanned at the client-specified scan rate.

**Initial Updates from Cache:** When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. A device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

## Device Properties — Tag Generation

The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the supporting driver) can be browsed from the clients.

If the target device supports its own local tag database, the driver reads the device's tag information and uses the data to generate tags within the server. If the device does not natively support named tags, the driver creates a list of tags based on driver-specific information. An example of these two conditions is as follows:

1. If a data acquisition system supports its own local tag database, the communications driver uses the tag names found in the device to build the server's tags.
2. If an Ethernet I/O system supports detection of its own available I/O module types, the communications driver automatically generates tags in the server that are based on the types of I/O modules plugged into the Ethernet I/O rack.

● **Note:** Automatic tag database generation's mode of operation is completely configurable. For more information, refer to the property descriptions below.

Property Groups	<input type="checkbox"/> <b>Tag Generation</b>	
General	On Duplicate Tag	Delete on Create
Scan Mode	Parent Group	
Timing	Allow Automatically Generated Subgroups	Enable
Auto-Demotion	Create	Create tags
<b>Tag Generation</b>		
Type		

### On Duplicate Tag

When automatic tag database generation is enabled, the server needs to know what to do with the tags that it may have previously added or with tags that have been added or modified after the communications driver since their original creation. This setting controls how the server handles OPC tags that were automatically generated and currently exist in the project. It also prevents automatically generated tags from accumulating in the server.

For example, if a user changes the I/O modules in the rack with the server configured to **Always Generate on Startup**, new tags would be added to the server every time the communications driver detected a new I/O module. If the old tags were not removed, many unused tags could accumulate in the server's tag space. The options are:

- **Delete on Create:** This option deletes any tags that were previously added to the tag space before any new tags are added. This is the default setting.
- **Overwrite as Necessary:** This option instructs the server to only remove the tags that the communications driver is replacing with new tags. Any tags that are not being overwritten remain in the server's tag space.
- **Do not Overwrite:** This option prevents the server from removing any tags that were previously generated or already existed in the server. The communications driver can only add tags that are completely new.
- **Do not Overwrite, Log Error:** This option has the same effect as the prior option, and also posts an error message to the server's Event Log when a tag overwrite would have occurred.

● **Note:** Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

**Parent Group:** This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically generated tags are added.

**Allow Automatically Generated Subgroups:** This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.

● **Note:** If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

**Create:** Initiates the creation of automatically generated OPC tags. If the device's configuration has been modified, **Create tags** forces the driver to reevaluate the device for possible tag changes. Its ability to be accessed from the System tags allows a client application to initiate tag database creation.

● **Note:** **Create tags** is disabled if the Configuration edits a project offline.

## Automatic Tag Database Generation

---

Automatic Tag Database Generation is a feature of the Hilscher Universal Driver that provides to import the SyCon Configuration Database into the OPC server. The specification of the SyCon Configuration file is made at the channel level; meaning, all devices under a channel are based on the same SyCon Configuration file.

Module definitions, module settings, configured I/O, and SyCon symbolic names are imported in tag database generation. In addition, diagnostics for Master and Slaves are generated. This import capability leaves the configuration in SyCon, with little configuration necessary in the server.

### How to Perform Automatic Tag Generation

To begin, ensure that devices have been defined under a channel. Remember that the device ID is the MAC ID in SyCon. To perform automatic tag generation, navigate to **Channel Properties | SyCon Database**. Click Synchronize tags to generate the tags.

Tags (I/O and Diagnostics) are generated for each device under the given channel. Wait until the process is complete before editing Device Properties, Channel Properties and the SyCon Configuration Database.

### Information Imported from Database

#### Master

Addressing Mode (Byte vs Word-Based Addressing)

#### Slave

Module Format (Byte, Word, or DWord Module)

Byte Swapping (Symbolic Name Swap Option)

### Information not Imported from Database

Device Names

Message Definitions (Explicit Messages, DPV1, etc)

String-type Symbolic Names

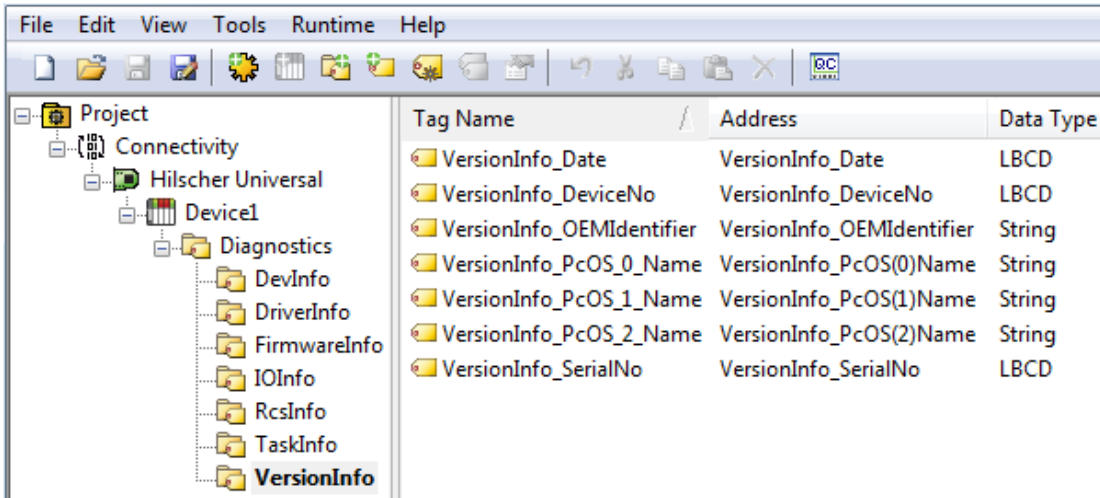
### Tags Generated in Server

#### Master

Diagnostic Tags



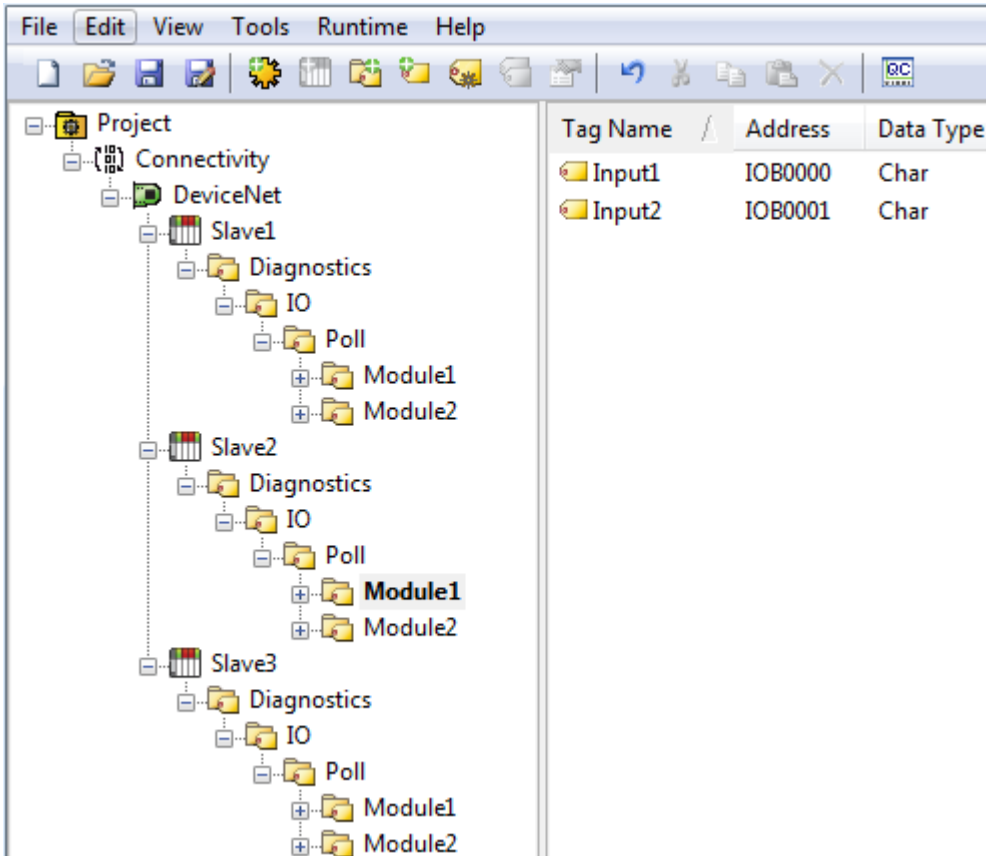
The image below is an example of the diagnostic tags available for a DeviceNet Master. The exact diagnostic tags generated depends on the bus as specified under **Channel Properties | Board Type**.



**Slave**

I/O Data Tags (SyCon Symbolic Names)  
 Diagnostic Tags (if applicable)

SyCon I/O tags defined in the SyCon Configuration Database are imported and a server tag generated for each. Also, if **Expanded SyCon Tag Import** is enabled, additional tags are generated based on both the SyCon I/O tags and the expansion settings selected. The image below displays the tags and tag groups generated for three DeviceNet devices with Expanded SyCon Tag Import enabled.



For information on Expanded SyCon I/O Tag Import, refer to [I/O Data References](#) and [Addressing Type](#).

## Device Properties — Timing

The device Timing properties allow the driver's response to error conditions to be tailored to fit the application's needs. In many cases, the environment requires changes to these properties for optimum performance. Factors such as electrically generated noise, modem delays, and poor physical connections can influence how many errors or timeouts a communications driver encounters. Timing properties are specific to each configured device.

Property Groups	<input type="checkbox"/> <b>Communication Timeouts</b>	
General	Connect Timeout (s)	3
Scan Mode	Request Timeout (ms)	5000
<b>Timing</b>	Retry Attempts	3
Auto-Demotion	<input type="checkbox"/> <b>Timing</b>	
	Inter-Request Delay (ms)	0

### Communications Timeouts

**Connect Timeout:** This property (which is used primarily by Ethernet based drivers) controls the amount of time required to establish a socket connection to a remote device. The device's connection time often takes longer than normal communications requests to that same device. The valid range is 1 to 30 seconds. The default is typically 3 seconds, but can vary depending on the driver's specific nature. If this setting is not supported by the driver, it is disabled.

**Note:** Due to the nature of UDP connections, the connection timeout setting is not applicable when communicating via UDP.

**Request Timeout:** This property specifies an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9,999,999 milliseconds (167.6667 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

**Retry Attempts:** This property specifies how many times the driver retries a communications request before considering the request to have failed and the device to be in error. The valid range is 1 to 10. The default is typically 3, but can vary depending on the driver's specific nature. The number of retries configured for an application depends largely on the communications environment. This property applies to both connection attempts and request attempts.

### Timing

**Inter-Request Delay:** This property specifies how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time reads and writes. This delay can be useful when dealing with devices with slow turnaround times and in cases where network load is a concern. Configuring a delay for a device affects communications with all other devices on the channel. It is recommended that users separate any device that requires an inter-request delay to a separate channel if possible. Other communications properties (such as communication serialization) can extend this delay. The valid range is 0 to 300,000 milliseconds; however, some drivers may limit the maximum value due to a function of their particular design. The default is 0, which indicates no delay between requests with the target device.

● **Note:** Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.

## Device Properties — Auto-Demotion

The Auto-Demotion properties can temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device offline for a specific time period, the driver can continue to optimize its communications with other devices on the same channel. After the time period has been reached, the driver re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.

Property Groups	[-] <b>Auto-Demotion</b>	
General	Demote on Failure	Enable
Scan Mode	Timeouts to Demote	3
Timing	Demotion Period (ms)	10000
<b>Auto-Demotion</b>	Discard Requests when Demoted	Disable

**Demote on Failure:** When enabled, the device is automatically taken off-scan until it is responding again.

● **Tip:** Determine when a device is off-scan by monitoring its demoted state using the `_AutoDemoted` system tag.

**Timeouts to Demote:** Specify how many successive cycles of request timeouts and retries occur before the device is placed off-scan. The valid range is 1 to 30 successive failures. The default is 3.

**Demotion Period:** Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

**Discard Requests when Demoted:** Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard writes; the server automatically fails any write request received from a client and does not post a message to the Event Log.

## Device Properties — Device Type

Property Groups	[-] <b>Type</b>	
General	Type	DeviceNet Slave
Auto-Demotion		DeviceNet Master
<b>Type</b>		DeviceNet Slave

**Type:** When applicable, this property defines the device's bus type and master/slave type. Supported device types include DeviceNet Master, DeviceNet Slave, Profibus-DP Master, and Profibus-DP Slave.

● **Note:** Diagnostics may be accessed from the device specified as the Master.

## Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8-bit value
Char	Signed 8-bit value
Word	Unsigned 16-bit value  bit 0 is the low bit bit 15 is the high bit
Short	Signed 16-bit value  bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
BCD	Two byte packed BCD  Value range is 0-9999. Behavior is undefined for values beyond this range.
DWord	Unsigned 32-bit value  bit 0 is the low bit bit 31 is the high bit
Long	Signed 32-bit value  bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
LBCD	Four byte packed BCD  Value range is 0-99999999. Behavior is undefined for values beyond this range.
Float	32-bit floating point value.  The driver interprets two consecutive 16 bit registers as a floating point value by making the second register the high word and the first register the low word.
String	Null-terminated ASCII string

## Address Descriptions

Select an addressing type from the list below for specific address descriptions.

### Process Image Address Descriptions

#### IEC Address Descriptions

● Process Image refers to addresses with syntax IOx and OOX. IEC refers to addresses with syntax Ix and Qx.

### Process Image Address Descriptions

Address mnemonic and offsets are based on the physical offset into the Master's Process Image Memory Map (I/O Data). Addresses are always byte-based, regardless of the addressing mode selected in SyCon's Master Settings. The default data types for dynamically defined tags are shown in **bold**.

● **Note:** The address ranges listed below are based on an 8K Dual-Port Memory.

Device Type	Range Data	Type	Access
Process Image Inputs	IOX0.b-IOX3583.b*	<b>Boolean</b>	Read Only
	.b is Bit 0-7 Byte Module		
	.b is Bit 0-15 Word Module		
	.b is Bit 0-31 DWord Module		
	IOB0-IOB3583	<b>Byte</b> , Char, String**	Read Only
	IOW0-IOW3582	<b>Word</b> , Short, BCD	Read Only
	IOD0-IOD3580	<b>DWord</b> , Long, LBCD, Float	Read Only
Process Image Outputs	OOX0.b-OOX3583.b*	<b>Boolean</b>	Write Only
	.b is Bit 0-7 Byte Module		
	.b is Bit 0-15 Word Module		
	.b is Bit 0-31 DWord Module		
	OOB0-OOB3583	<b>Byte</b> , Char, String**	Write Only
	OOW0-OOW3582	<b>Word</b> , Short, BCD	Write Only
	OOD0-OOD3580	<b>DWord</b> , Long, LBCD, Float	Write Only

\*These memory types / subtypes do not support arrays.

\*\*Byte memory types (IOB) support Strings. The syntax for strings is *<address>.<length>* where 0 < length <= 246.

#### ● Notes :

1. All offsets for memory types IO and OO represent a byte starting location within the specified memory type.

2. Use caution when modifying Word, Short, DWord and Long types. For I and Q memory types, addresses may overlap depending on the module format and addressing type. It is recommended that these memory types be used so that overlapping does not occur.

• For information on referencing of Process Image data, refer to [Module Format vs. Byte/Word Addressing](#).

## Arrays

All memory types support arrays, excepting those marked with an asterisk (\*). The syntax below is valid for declaring an array. If no rows are specified, row count of 1 is assumed.

`<address>[rows][cols]`

For Word, Short and BCD arrays, the base address + (rows \* cols \* 2) cannot exceed 247. The array's elements are words and are located on a word boundary. For example, IOW0[4] will return IOW0, IOW2, IOW4, and IOW6 for both Byte and Word Addressing.

For Float, DWord, Long and Long BCD arrays, the base address + (rows \* cols \* 4) cannot exceed 247. The array's elements are DWords and are located on a DWord boundary. For example, IOD0[4] will return IOD0, IOD4, IOD8, IOD12 for both Byte and Word Addressing.

For all arrays, the total number of bytes being requested cannot exceed the internal block size of 247 bytes.

## Byte Swapping

Bytes can be swapped for 16 bit (Word, Short and BCD) and 32 bit (DWord, Long, Float and LBCD) data by appending an 'S' to the end of an address reference:

`<address>S`

For arrays (each element is Byte swapped):

`<address>S [rows][cols]`

Below are examples to illustrate how Bytes are swapped for 16 and 32 bit data for both Little and Big Endian. Byte-ordering is unaffected by the Addressing Mode.

### 16-Bit Data

In the example below, an analog sensor maps to Offset 0. Sensor value=0x1234 (hex) == 4660 (dec).

#### Little Endian (LSB-MSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x34	IOB0=0x34	IOW0=0x1234	IOD0=0x00001234
1	0x12	IOB1=0x12		
2	0x00	IOB2=0x00		
3	0x00	IOB3=0x00		

#### Little Endian (LSB-MSB) - Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x34	IOB0=0x34	IOW0=0x3412	IOD0=0x34120000
1	0x12	IOB1=0x12		
2	0x00	IOB2=0x00		
3	0x00	IOB3=0x00		

#### Big Endian (MSB-LSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IOB0=0x12	IOW0=0x3412	IOD0=0x00003412
1	0x34	IOB1=0x34		
2	0x00	IOB2=0x00		
3	0x00	IOB3=0x00		

#### Big Endian (MSB-LSB) - Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IOB0=0x12	IOW0=0x1234	IOD0=0x12340000
1	0x34	IOB1=0x34		
2	0x00	IOB2=0x00		
3	0x00	IOB3=0x00		

#### 32-Bit Data

In the example below, an analog sensor maps to Offset 0. Sensor value=0x12345678 (hex) == 305,419,896 (dec).

#### Little Endian (LSB-MSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x78	IOB0=0x78	IOW0=0x5678	IOD0=0x12345678
1	0x56	IOB1=0x56		
2	0x34	IOB2=0x34	IOW2=0x1234	
3	0x12	IOB3=0x12		

#### Little Endian (LSB-MSB) - Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x78	IOB0=0x78	IOW0=0x7856	IOD0=0x78563412
1	0x56	IOB1=0x56		
2	0x34	IOB2=0x34	IOW2=0x3412	
3	0x12	IOB3=0x12		

#### Big Endian (MSB-LSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IOB0=0x12	IOW0=0x3412	IOD0=0x78563412
1	0x34	IOB1=0x34		
2	0x56	IOB2=0x56	IOW2=0x7856	
3	0x78	IOB3=0x78		

### Big Endian (MSB-LSB) - Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IOB0=0x12	IOW0=0x1234	IOD0=0x12345678
1	0x34	IOB1=0x34		
2	0x56	IOB2=0x56	IOW2=0x5678	
3	0x78	IOB3=0x78		

### Notes:

1. Caution must be exercised when referencing overlapped memory. For example, IW1S will corrupt IW0S (and vice-versa). Overlapping references is not recommended.
2. SyCon allows for swapping via Symbolic Names under Long and Word Details.

### Module Format vs. Byte/Word Addressing

Module Format refers to the width of the Module configured in SyCon. Supported formats include Byte, Word and DWord. Addressing Modes below refer to the Addressing Mode under Master Settings in SyCon. Byte 0 - Byte n refer to the Byte Offsets in the Process Image. Regions highlighted in exemplify the Bytes involved in a reference at Offset 0 for the given memory type (i.e., IOD0 will contain Byte 0-Byte 3). They also illustrates how Words and DWords can overlap. Exercise caution when referencing overlapped Words and DWords.

### Byte Module (8-Bit Module Data)

For Byte Modules, Byte memory is treated as a Byte. There are no N/A references.

### Byte Addressing

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-7	IOX1.0-7	IOX2.0-7	IOX3.0-7	IOX4.0-7
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	IOW1	IOW2	IOW3	IOW4
IOD0	IOD1	IOD2	IOD3	IOD4

Table x: Byte Module Byte PI Addressing

### Word Addressing

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-7	IOX1.0-7	IOX2.0-7	IOX3.0-7	IOX4.0-7
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	IOW1	IOW2	IOW3	IOW4
IOD0	IOD1	IOD2	IOD3	IOD4



**Table x: Byte Module Word PI Addressing****Word Module (16 bit Module Data)**

For Word Modules, Byte memory is treated as a Word with the exception of IOB. N/A references are due to this Word alignment for the module.

**Byte Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-15	N/A	IOX2.0-15	N/A	IOX4.0-15
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	N/A	IOW2	N/A	IOW4
IOD0	N/A	IOD2	N/A	IOD4

**Table x: Word Module Byte PI Addressing****Word Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-15	N/A	IOX2.0-15	N/A	IOX4.0-15
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	N/A	IOW2	N/A	IOW4
IOD0	N/A	IOD2	N/A	IOD4

**Table x: Word Module Word PI Addressing****DWord Module (32 bit Module Data)**

For DWord Modules, Byte memory is treated as a DWord with the exception of IOB and IOW. N/A references are due to this DWord alignment for the module.

**Byte Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-31	N/A	N/A	N/A	IOX4.0-31
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	N/A	IOW2	N/A	IOW4
IOD0	N/A	N/A	N/A	IOD4

**Table x: DWord Module Byte PI Addressing****Word Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IOX0.0-31	N/A	N/A	N/A	IOX4.0-31
IOB0	IOB1	IOB2	IOB3	IOB4
IOW0	N/A	IOW2	N/A	IOW4
IOD0	N/A	N/A	N/A	IOD4

**Table x: DWord Module Word PI Addressing**

## IEC Address Descriptions

Address mnemonic and offsets are based on standard Siemens addressing (IB, IW, ID). Addresses are byte-based or word-based, depending on the addressing mode selected in SyCon's Master Settings. The default data types for dynamically defined tags are shown in **bold**.

**Note:** The address ranges listed below are based on an 8K Dual-Port Memory.

Device Type	Range Data	Type	Access
Process Image Inputs	IX0.b-IX3583.b*	<b>Boolean</b>	Read Only
	.b is Bit 0-7 Byte Module		
	.b is Bit 0-15 Word Module		
	.b is Bit 0-31 DWord Module		
	IB0-IB3583	<b>Byte</b> , Char, String**	Read Only
	IW0-IW3582	<b>Word</b> , Short, BCD	Read Only
	ID0-ID3580	<b>DWord</b> , Long, LBCD, Float	Read Only
Process Image Outputs	QX0.b-QX3583.b*	<b>Boolean</b>	Write Only
	.b is Bit 0-7 Byte Module		
	.b is Bit 0-15 Word Module		
	.b is Bit 0-31 DWord Module		
	QB0-QB3583	<b>Byte</b> , Char, String**	Write Only
	QW0-QW3582	<b>Word</b> , Short, BCD	Write Only
	QD0-QD3580	<b>DWord</b> , Long, LBCD, Float	Write Only

\*These memory types / subtypes do not support arrays.

\*\*Byte memory types (IB) support Strings. The syntax for strings is `<address>.<length>` where  $0 < \text{length} \leq 246$ .

**Note:** All offsets for memory types I and Q represent a byte starting location within the specified memory type.

Use caution when modifying Word, Short, DWord and Long types. For I and Q memory types, addresses may overlap depending on the module format and addressing type. It is recommended that these memory types be used so that overlapping does not occur.

For information on the proper referencing of Process Image data, refer to [Module Format vs. Byte / Word Addressing](#).

### Arrays

All memory types support arrays, excepting those marked with an asterisk (\*). The syntax below is valid for declaring an array. If no rows are specified, row count of 1 is assumed.

`<address>[rows][cols]`

For Word, Short and BCD arrays, the base address + (rows \* cols \* 2) cannot exceed 247. The array's elements are words and are located on word boundaries. For example, IW0[4] would return IW0, IW2, IW4, and IW6 assuming Byte Addressing, IW0, IW1, IW2, IW3 (assuming Word Addressing).

For Float, DWord, Long and Long BCD arrays, the base address + (rows \* cols \* 4) cannot exceed 247. The array's elements are DWords and are located on DWord boundaries. For example, ID0[4] will return ID0, ID4, ID8, ID12 (assuming Byte Addressing) and ID0, ID2, ID4, ID6 (assuming Word Addressing).

For all arrays, the total number of bytes being requested cannot exceed the internal block size of 247 bytes.

### Byte Swapping

Bytes can be swapped for 16 bit (Word, Short and BCD) and 32 bit (DWord, Long, Float and LBCD) data by appending an 'S' to the end of an address reference:

`<address>S`

For arrays (each element is Byte swapped):

`<address>S [rows][cols]`

The examples below illustrate how Bytes are swapped for 16 and 32 bit data for both Little and Big Endian. Byte-ordering is unaffected by the addressing ,ode.

#### 16-Bit Data

In the example below, an analog sensor maps to Offset 0. Sensor value=0x1234 (hex) == 4660 (dec).

#### Little Endian (LSB-MSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x34	IB0=0x34	IW0=0x1234	ID0=0x00001234
1	0x12	IB1=0x12		
2	0x00	IB2=0x00		
3	0x00	IB3=0x00		

#### Little Endian (LSB-MSB) - Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x34	IB0=0x34	IW0=0x3412	ID0=0x34120000
1	0x12	IB1=0x12		
2	0x00	IB2=0x00		
3	0x00	IB3=0x00		

#### Big Endian (MSB-LSB) - No Swap

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IB0=0x12	IW0=0x3412	ID0=0x00003412
1	0x34	IB1=0x34		

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
2	0x00	IB2=0x00		
3	0x00	IB3=0x00		

**Big Endian (MSB-LSB) - Swap**

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IB0=0x12	IW0=0x1234	ID0=0x12340000
1	0x34	IB1=0x34		
2	0x00	IB2=0x00		
3	0x00	IB3=0x00		

**32-Bit Data**

In the example below, an analog sensor maps to Offset 0. Sensor value=0x12345678 (hex) == 305,419,896 (dec).

**Little Endian (LSB-MSB) - No Swap**

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x78	IB0=0x78	IW0=0x5678	ID0=0x12345678
1	0x56	IB1=0x56		
2	0x34	IB2=0x34	IW1=0x1234	
3	0x12	IB3=0x12		

**Little Endian (LSB-MSB) - Swap**

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x78	IB0=0x78	IW0=0x7856	ID0=0x78563412
1	0x56	IB1=0x56		
2	0x34	IB2=0x34	IW1=0x3412	
3	0x12	IB3=0x12		

**Big Endian (MSB-LSB) - No Swap**

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IB0=0x12	IW0=0x3412	ID0=0x78563412
1	0x34	IB1=0x34		
2	0x56	IB2=0x56	IW1=0x7856	
3	0x78	IB3=0x78		

**Big Endian (MSB-LSB) - Swap**

DPM Byte Offset	Data	Byte Reference	Word Reference	DWord Reference
0	0x12	IB0=0x12	IW0=0x1234	ID0=0x12345678
1	0x34	IB1=0x34		
2	0x56	IB2=0x56	IW1=0x5678	
3	0x78	IB3=0x78		

⚠ Caution must be exercised when referencing overlapped memory. For example, IW1S will corrupt IW0S (and vice-versa). Overlapping references is not recommended.

● **Note:** SyCon allows for swapping via symbolic names under Long and Word Details

### Module Format vs. Byte / Word Addressing

Module Format refers to the width of the Module configured in SyCon. Supported formats include Byte, Word and DWord. The addressing modes below refer to SyCon's Addressing Mode settings. Byte 0-Byte n refer to the Byte Offsets in the Process Image. Regions highlighted exemplify the Bytes involved in a reference at Offset 0 for the given memory type (ie ID0 will contain Byte 0-Byte 3). They also illustrates how Words and DWords can overlap. Exercise caution when referencing overlapped Words and DWords.

### Byte Module (8-Bit Module Data)

For Byte Modules, Byte memory is treated as a Byte. There are no N/A references.

#### Byte Addressing

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-7	IX1.0-7	IX2.0-7	IX3.0-7	IX4.0-7
IB0	IB1	IB2	IB3	IB4
IW0	IW1	IW2	IW3	IW4
ID0	ID1	ID2	ID3	ID4

Table x: Byte Module Byte IEC Addressing

#### Word Addressing

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-7	N/A	IX1.0-7	N/A	IX2.0-7
IB0	N/A	IB1	N/A	IB2
IW0	N/A	IW1	N/A	IW2
ID0	N/A	ID1	N/A	ID2

Table x: Byte Module Word IEC Addressing

### Word Module (16-Bit Module Data)

For Word Modules, Byte memory is treated as a Word (with the exception of IOB). N/A references are due to the Word alignment for the module.

#### Byte Addressing

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-15	N/A	IX2.0-15	N/A	IX4.0-15

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IB0	IB1	IB2	IB3	IB4
IW0	N/A	IW2	N/A	IW4
ID0	N/A	ID2	N/A	ID4

Table x: Word Module Byte IEC Addressing

**Word Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-15	N/A	IX1.0-15	N/A	IX2.0-15
IB0	N/A	IB1	N/A	IB2
IW0	N/A	IW1	N/A	IW2
ID0	N/A	ID1	N/A	ID2

Table x: Word Module Word IEC Addressing

**DWord Module (32-Bit Module Data)**

For DWord Modules, Byte memory is treated as a DWord (with the exception of IOB and IOW). N/A references are due to the DWord alignment for the module.

**Byte Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-31	N/A	N/A	N/A	IX4.0-31
IB0	IB1	IB2	IB3	IB4
IW0	N/A	IW2	N/A	IW4
ID0	N/A	N/A	N/A	ID4

Table x: DWord Module Byte IEC Addressing

**Word Addressing**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
IX0.0-31	N/A	N/A	N/A	IX2.0-31
IB0	N/A	IB1	N/A	IB2
IW0	N/A	IW1	N/A	IW2
ID0	N/A	N/A	N/A	ID2

Table x: DWord Module Word IEC Addressing

## Error Descriptions

---

The following error/warning messages may be generated. Click on the link for a description of the message.

### Address Validation

#### [Missing address](#)

[Device address '<address>' contains a syntax error](#)

[Address '<address>' is out of range for the specified device or register](#)

[Data Type '<type>' is not valid for device address '<address>'](#)

[Device address '<address>' is Read Only](#)

[Array size is out of range for address '<address>'](#)

[Array support is not available for the specified address: '<address>'](#)

### Driver Error Messages

[Unable to load '<dll>'](#)

[Unable to import from '<dll>'](#)

[DevOpenDriver \(\) failed with error code '<code>'](#)

[Memory allocation error](#)

### Device Status Messages

[Device '<device name>' is not responding](#)

[Unable to read device info data in area '<area>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read '<block size>' device info bytes in area '<area>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read task state data in task '<task num>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read '<block size>' task state bytes in task '<task num>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'](#)

[Unable to read tag '<name>': msg.b <command>, msg.device\\_adr <Device ID>](#)

[Unable to read '<block size>' message bytes: msg.b <command>, msg.device\\_adr <Device ID>...](#)

[Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics \[Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'\]](#)

[Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics \[Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'\]](#)

[Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics \[Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>'\]](#)

Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']

Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']

Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']

### Automatic Tag Database Generation Messages

The file is not a valid Sycon database or may be corrupt

Auto tag database generation cannot be performed while the driver is processing tags

Board Type for Board '<board number>' does not match the actual board installed. Verify Board Type and/or Board Selection

Board Type for Board '<board number>' does not match the Slave Type for one or more Slaves configured. Delete or edit Slaves accordingly

'dbm32.dll' is not loaded and is required for auto tag generation. Verify SyCon is installed

### Error Codes

#### CIF Device Driver Errors

Error Code	Source	Description
-1	CIF Driver	The communication board is not initialized by the driver.  -Check the driver configuration. -Driver function used without calling DevOpenDriver () first.
-2	CIF Driver	Error in internal 'Init State'.
-3	CIF Driver	Error in internal 'Read State'.
-4	CIF Driver	Command on this channel is active.
-5	CIF Driver	Unknown parameter in function occurred.
-6	CIF Driver	Version is incompatible. The device driver version does not correspond to the driver DLL version. From version V1.200 the internal command structure between DLL and driver has changed. Make sure to use the same version of the device driver and the driver DLL.
-10	Device	Dual port memory RAM is not accessible / no hardware found. This error occurs when the driver is not able to read or write to the Dual port memory.  -Check the BIOS setting of the PC.  Memory address conflict with other PC components, try another memory address.



Error Code	Source	Description
		-Check the driver configuration for this board -Check the jumper settings of the board.
-11	Device	Not ready (RDY flag=Ready flag failed). Board is not ready. This could be a hardware malfunction or another program writes inadmissible to the dual port memory.
-12	Device	Not running (RUN flag=Running flag failed). The board is ready but not all tasks are running because of an initialization error.  -No database is loaded into the device or an invalid parameter has been set so that a task cannot initialize.
-13	Device	Watchdog test failed.
-14	Device	Signals wrong Operating System version. No license code found on the communication board.  -Device has no license for the used operating system or customer software. -No firmware or no database on the device is loaded.
-15	Device	Error in dual port memory flags.
-16	Device	Send mailbox is full.
-17	Device	Function PutMessage timeout.  -If using an interrupt, check the interrupt on the device and in driver setup. These settings have to be the same. Is an interrupt on the board set? Is the right interrupt set? The interrupt could already be used by another PC component. -If using polling mode, make sure that no interrupt is set on the board and that polling is set in the driver setup. The settings have to be the same. -Device internal segment buffer full. DevSetHostState not called.
-18	Device	Function GetMessage timeout.  -If using an interrupt, check the interrupt on the device and in driver setup. These settings have to be the same. Is an interrupt on the board set? Is the right interrupt set? The interrupt could already be used by another PC component. -If using polling mode, make sure that no interrupt is set on the board and that polling is set in the driver setup. The settings have to be the same.
-19	Device	No message available.
-20	Device	Reset command timeout.  -The board is ready but not all tasks are running because of an initialization error. -No database is loaded into the device. -If using an interrupt, check the interrupt on the device and in driver setup. These settings have to be the same. Is an interrupt on the board set? Is the right interrupt set? The interrupt could already be used by another PC component. -If using polling mode, make sure that no interrupt is set on the board and that polling is set in the driver setup. The settings have to be the same.
-21	Device	COM flag not set. The device cannot reach communication state.

Error Code	Source	Description
		-Device not connected to the fieldbus. -No station found on the fieldbus. -Wrong configuration on the device.
-22	Device	I/O data exchange failed.
-23	Device	I/O data exchange timeout.  -If using an interrupt, check the interrupt on the device and in driver setup. These settings have to be the same. Is an interrupt on the board set? Is the right interrupt set? The interrupt could already be used by another PC component. -If using polling mode, then make sure that no interrupt is set on the board and that polling is set in the driver setup. The settings have to be the same.
-24	Device	I/O data mode unknown.
-25	Device	Function call failed.
-26	Device	Dual port memory size differs from configuration.
-27	Device	State mode unknown.
-30	User	Driver not opened (device driver not loaded).  -Device driver not installed. -Wrong parameters in the driver configuration.
-31	User	Can't connect with device board.
-32	User	Board not initialized.  -DevInitBoard () not called.
-33	User	IOCTRL function failed.  -Make sure to use a device driver and DLL with the same version.
-34	User	Parameter DeviceNumber invalid.
-35	User	Parameter InfoArea unknown.
-36	User	Parameter Number invalid.
-37	User	Parameter Mode invalid.
-38	User	NULL pointer assignment.
-39	User	Message buffer too short.
-40	User	Size parameter invalid.
-42	User	Size parameter with zero length.
-43	User	Size parameter too long.
-44	User	Device address null pointer.
-45	User	Pointer to buffer is a null pointer.
-46	User	SendSize parameter too long.
-47	User	ReceiveSize parameter too long.
-48	User	Pointer to send buffer is a null pointer.
-49	User	Pointer to receive buffer is a null pointer.

---

## Missing address

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically has no length.

**Solution:**

Re-enter the address in the client application.

---

## Device address '<address>' contains a syntax error

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically contains one or more invalid characters.

**Solution:**

Re-enter the address in the client application.

---

## Address '<address>' is out of range for the specified device or register

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically references a location that is beyond the range of supported locations for the device.

**Solution:**

Verify the address is correct; if it is not, re-enter it in the client application.

---

## Data Type '<type>' is not valid for device address '<address>'

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically has been assigned an invalid data type.

**Solution:**

Modify the requested data type in the client application.

---

## Device address '<address>' is Read Only

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically has a requested access mode that is not compatible with what the device supports for that address.

**Solution:**

Change the access mode in the client application.

---

**Array size is out of range for address '<address>'**

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically is requesting an array size that is too large for the address type or block size of the driver.

**Solution:**

Re-enter the address in the client application to specify a smaller value for the array or a different starting point.

---

**Array Support is not available for the specified address: '<address>'**

---

**Error Type:**

Warning

**Possible Cause:**

A tag address that has been specified statically contains an array reference for an address type that doesn't support arrays.

**Solution:**

Re-enter the address in the client application to remove the array reference or correct the address type.

---

**Unable to load '<dll>'**

---

**Error Type:**

Serious

**Possible Cause:**

A software component necessary to communicate with the Hilscher card or SyCon configuration database cannot be loaded.

**Solution:**

Verify that the latest version of SyCon is installed on the same machine as the OPC server and then try again.

---

**Unable to import from '<dll>'**

---

**Error Type:**

Serious

**Possible Cause:**

The interface necessary to communicate with the Hilscher card or SyCon configuration database, cannot be loaded from <dll>.

**Solution:**

Verify that the latest version of SyCon is installed on the same machine as the OPC server and try again.

---

**DevOpenDriver () failed with error code '<code>'****Error Type:**

Serious

**Possible Cause:**

Unable to load the device drivers necessary for card communications.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Memory allocation error****Error Type:**

Serious

**Possible Cause:**

Memory required to driver operations could not be allocated.

**Solution:**

Close any unused applications and / or increase the amount of virtual memory. Then, try again.

---

**Device '<device name>' is not responding****Error Type:**

Warning

**Result:****If the tag was being read:**

- If tag is a block tag, the entire block is invalidated. All tags within that block are invalidated.
- If tag is an array tag or string tag, just this tag is invalidated.

**If the tag was being written:**

- Write operation for the given tag will not take place.

**Possible Cause:**

1. The connection between the device and the Host PC is broken.
2. Device CPU work load is too high.

3. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

**Solution:**

1. Verify the cabling between the PC and the PLC device.
2. If this error occurs frequently, decrease the tag group scan rate to reduce the work load on the PLC CPU.
3. Increase the Request Timeout setting so that the entire response can be handled.

---

**Unable to read device info data in area '<area>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

Device info data contains diagnostics information for most Master cards. Some device info data is read individually while other device info data are structured and read as a block. This error pertains to the former. Read access to this data failed.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read '<block size>' device info bytes in area '<area>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

Device info data contains diagnostics information for most Master cards. Some device info are read individually while other device info data are structured and read as a block. This error pertains to the latter. Read access to this data failed.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read task state data in task '<task num>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

Task state data contains diagnostics information for some Slave cards. Some task data is read individually while other device info data are structured and read as a block. This error pertains to the former. Read access to this data failed.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read '<block size>' task state bytes in task '<task num>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

Task state data contains diagnostics information for some Slave cards. Some task data is read individually while other device info data are structured and read as a block. This error pertains to the latter. Read access to this data failed.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read the data at offset '<address>' in '<board>'s Input image. This data location corresponds to Input data mapped for device '<device>'. This error pertains to all string and array I/O tags.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read '<block size>' bytes of data, starting at offset '<address>', in '<board>'s Input image. These data locations correspond to Input data mapped for device '<device>'. This error pertains to all non-string and non-array I/O tags.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

A write failed to offset '<address>' in '<board>'s Output image, corresponding to Output data mapped to device '<device>'.

**Solution:**

Refer to the Error Codes table for specific information.

**See Also:**

[Error Codes](#)

---

**Unable to read tag '<name>': msg.b=<command>, msg.device\_adr=<Device ID>...**

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read tag '<name>' because the message associated with '<name>' failed or the response from the message was invalid. The message command and destination device are listed as <command> and <Device ID> respectively. Applicable for diagnostics only.

- Refer to the Error Codes table if the following message is received:  
"...Get message failed on Board '<board>' with Error Code '<code>'"
- Either the destination for the message was invalid or unexpected data was received the following message is received:  
"...Response from Board '<board>' contains a framing error"



**Solution:**

Re-download the configuration database in SyCon.

**See Also:**

[Error Codes](#)

### Unable to read '<block size>' message bytes: msg.b=<command>, msg.device\_adr=<Device ID>...

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read <block size> bytes of data requested in message <command>, from device <Device ID> because the message failed or the response from the message was invalid. The message command and destination device are listed as <command> and <Device ID> respectively. Applicable for diagnostics only.

- Refer to the Error Codes table if the following message is received:  
"...Get message failed on Board '<board>' with Error Code '<code>'"
- Either the destination for the message was invalid or unexpected data was received if the following message is received:  
"...Response from Board '<board>' contains a framing error"

**Solution:**

Re-download the configuration database in SyCon.

**See Also:**

[Error Codes](#)

### Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>']

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read the data at offset '<address>' in '<board>'s Input image. This data location corresponds to Input data mapped for device '<device>'. This error pertains to all non-string and non-array I/O tags.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

### Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>']

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read '<block size>' bytes of data, starting at offset '<address>', in '<board>'s Input image. These data locations correspond to Input data mapped for device '<device>'. This error pertains to all non-string and non-array I/O tags.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

---

**Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics [Global Bits='<Global Bits>', Node='<Remote Address>', Code='<Error Event>']**

---

**Error Type:**

Warning

**Possible Cause:**

A write failed to offset '<address>' in '<board>'s Output image, corresponding to Output data mapped to device '<device>'.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

---

**Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']**

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read the data at offset '<address>' in '<board>'s Input image. This data location corresponds to Input data mapped for device '<device>'. This error pertains to all non-string and non-array I/O tags.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

---

**Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']**

---

**Error Type:**

Warning

**Possible Cause:**

The driver was unable to read '<block size>' bytes of data, starting at offset '<address>', in '<board>'s Input image. These data locations correspond to Input data mapped for device '<device>'. This error pertains to all non-string and non-array I/O tags.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

---

**Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics [Global Bits='<Global Bits>', Node='<Device Address>', Code='<Error Event>']**

---

**Error Type:**

Warning

**Possible Cause:**

A write failed to offset '<address>' in '<board>'s Output image, corresponding to Output data mapped to device '<device>'.

**Solution:**

Please contact Technical Support for information specific to the returned DPM Diagnostics information.

---

**The file is not a valid Sycon database or may be corrupt**

---

**Error Type:**

Warning

**Possible Cause:**

1. The file is not a valid Sycon database.
2. The file is corrupt.

**Solution:**

1. Ensure that the file is a valid Sycon database.
2. Ensure that the file is not corrupt.
3. Attempt using a new and valid file.

---

**Auto tag database generation cannot be performed while the driver is processing tags**

---

**Error Type:**

Warning

**Possible Cause:**

Automatic tag database generation was attempted while the driver was processing tags.

**Solution:**

Ensure that the driver is not processing tags and then reattempt automatic tag database generation.

---

**Board Type for Board '<board number>' does not match the actual board installed. Verify Board Type and/or Board Selection**

---

**Error Type:**

Warning

**Possible Cause:**

The Board Type that is being used does not match the board that is being installed.

**Solution:**

Verify the Board Type and/or the Board Selection.

---

**Board Type for Board '<board number>' does not match the Slave Type for one or more Slaves configured. Delete or edit Slaves accordingly**

---

**Error Type:**

Warning

**Possible Cause:**

The Board Type does not match the Slave Type for one or more of the slaves being configured.

**Solution:**

Edit or delete the slaves as necessary in order to ensure that the Board Type matches the Slave Type.

---

**dbm32.dll is not loaded and is required for auto tag generation. Verify SyCon is installed**

---

**Error Type:**

Warning

**Possible Cause:**

'dbm32.dll' is not loaded.

**Solution:**

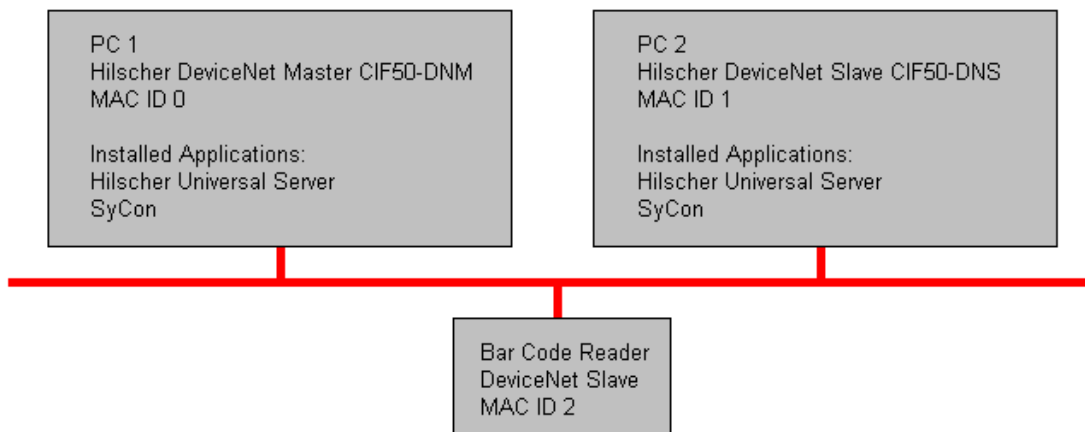
1. Load 'dbm32.dll' and then reattempt automatic tag database generation.
2. Verify that Sycon is installed.

## Appendix: Slave Board Configuration

Once configured, a local Hilscher Slave board can communicate both locally and with the Master. For information on configuration (and local / master communication) refer to the instructions below.

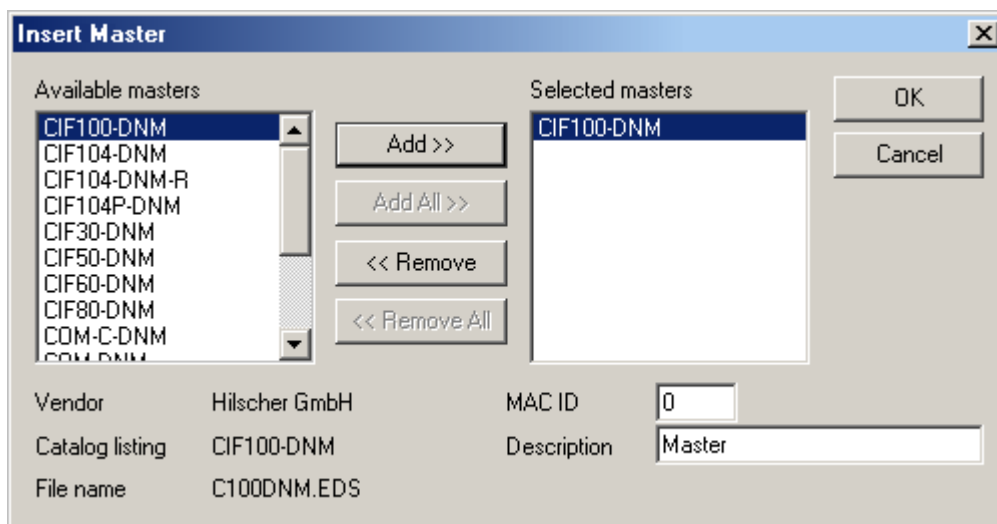
● **Note** : This tutorial uses DeviceNet as the example network, but the same steps may be applied to Profibus.

● **Important**: Two SyCon configurations are required in order to communicate to the Slave board locally and from the Master. One must be from the Master's perspective and one must be from the Slave's perspective. In the example below, the Master board is in PC 1 and the Slave board is in PC 2. The Slave board is configured before the Master.



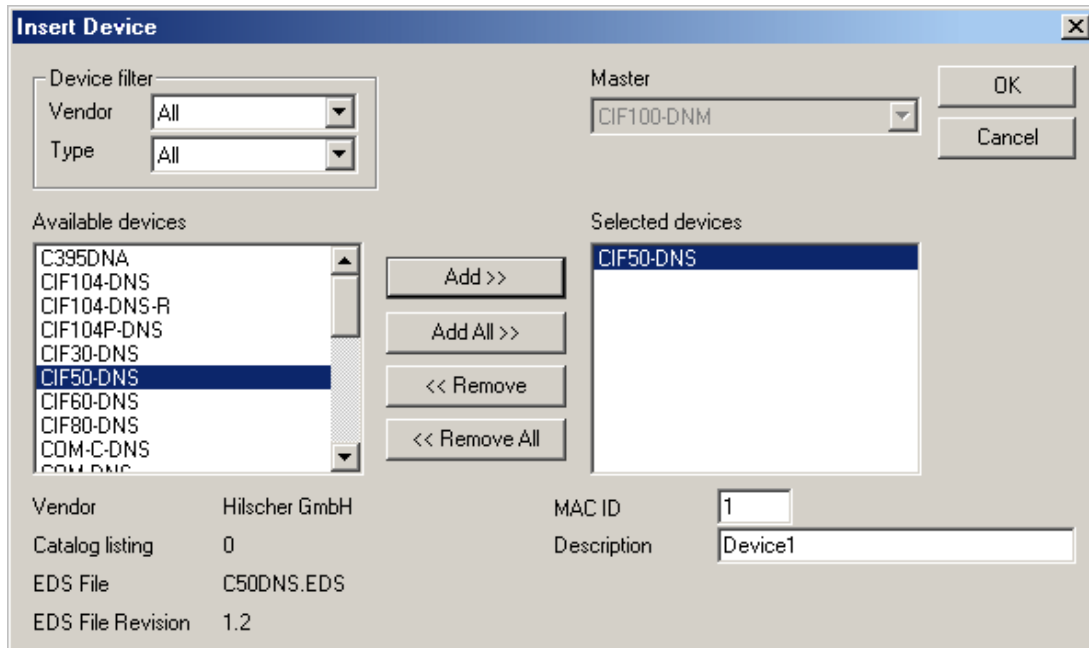
### PC 2 SyCon Configuration (Slave)

1. To start, open an empty SyCon project and insert a "dummy" Master. The master chosen is irrelevant since the configuration is downloaded to the Slave.

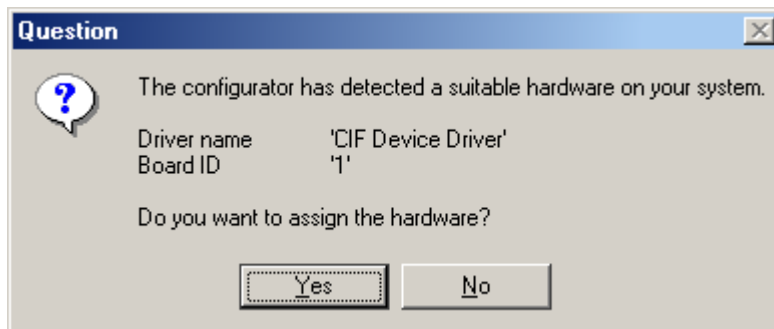


● **Note**: When prompted, do not assign the hardware if the Master board is in the same machine as the Slave board.

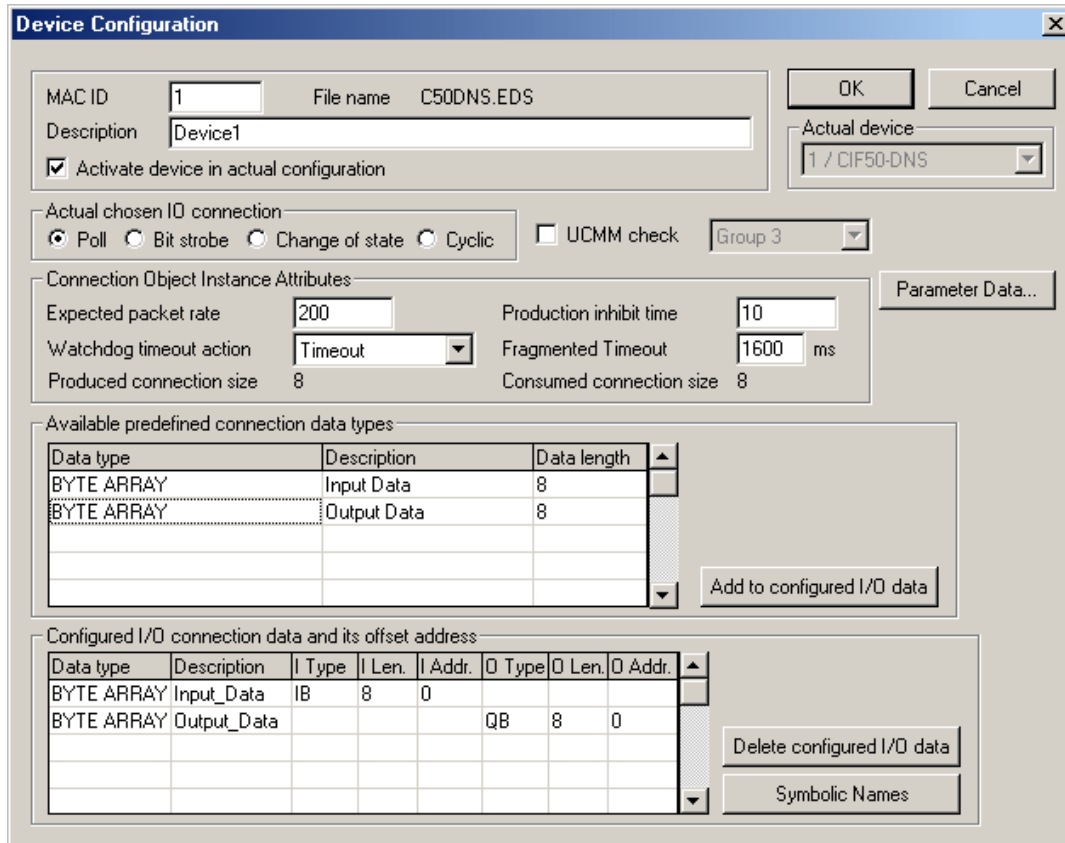
- Next, insert the Slave and select a unique MAC ID.



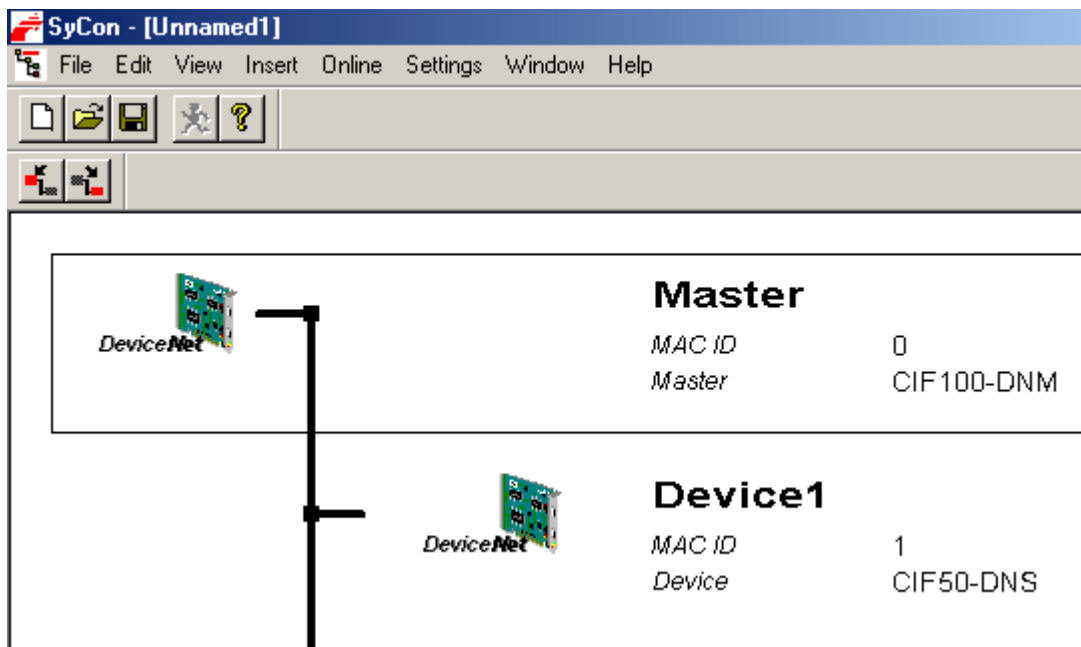
- Assign the MAC ID to the local Slave board.



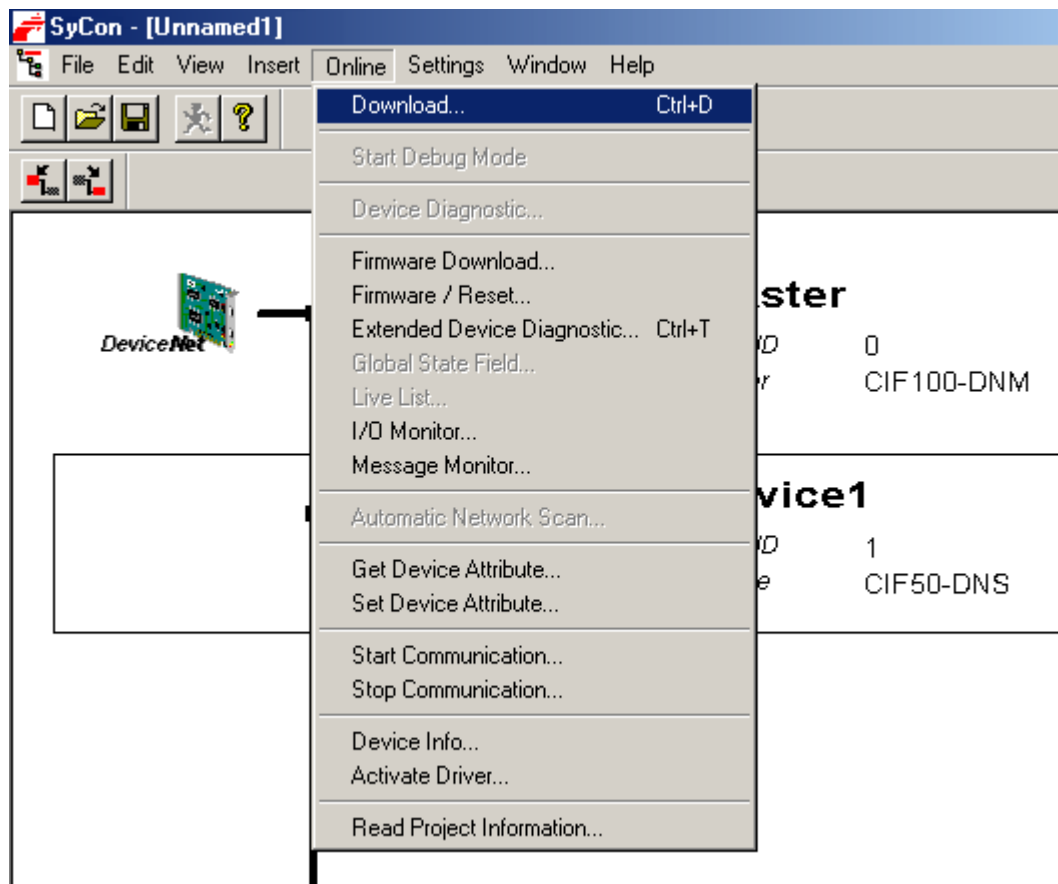
- Then, configure the I/O.



**Note:** The network should appear as shown below.



5. Save and then download the configuration to the Slave board by clicking **Online | Download**. This is the configuration that is imported into the server later.

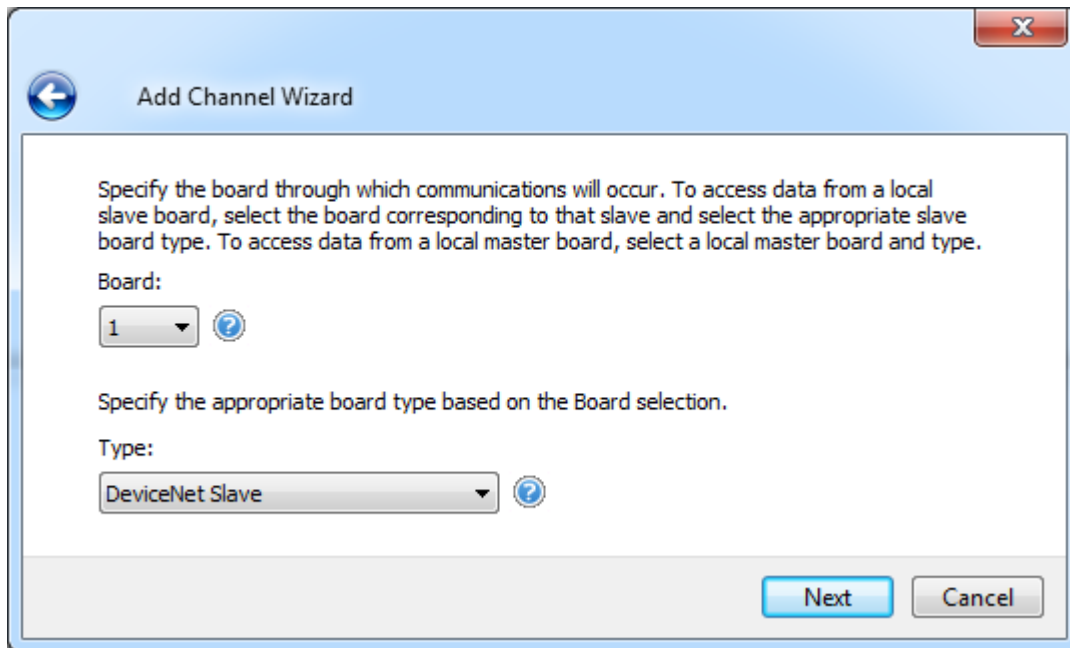


### PC 2 Server Configuration (Slave)

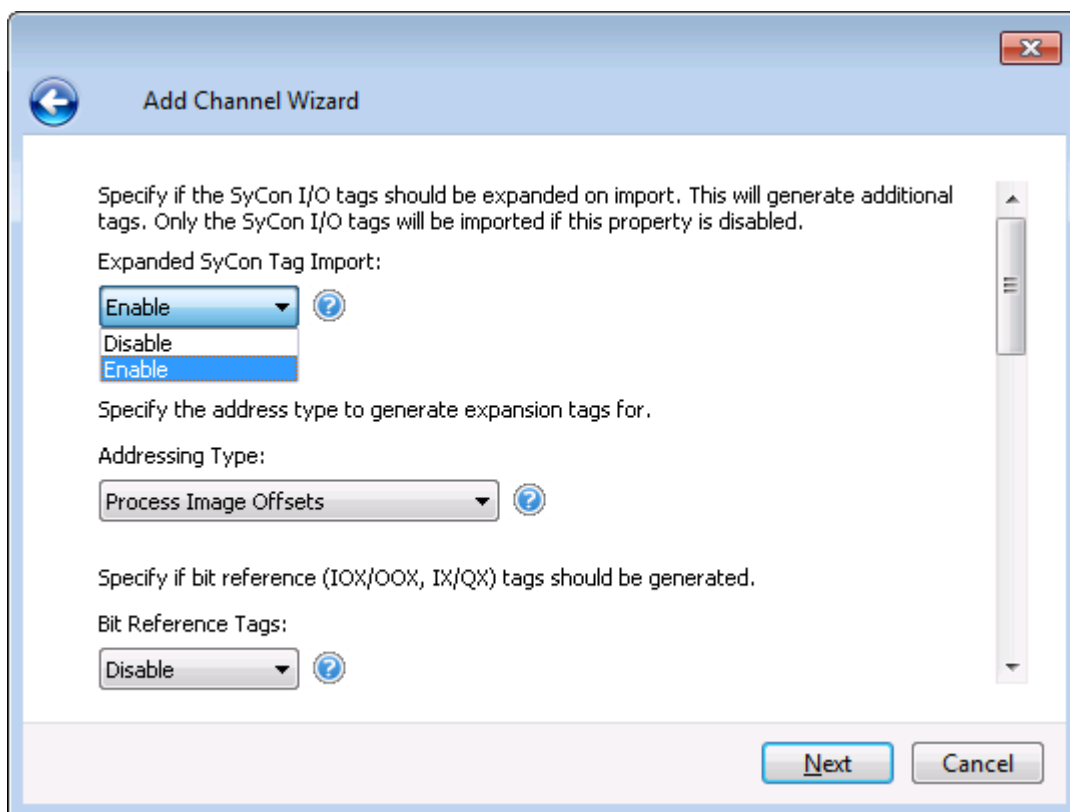
Once the Slave board is configured, it can be accessed locally using the Hilscher Universal Driver.

1. In the server, create a new channel. In the **Board Selection** dialog, choose the location of the board in the PC and then choose **DeviceNet Slave** as a board type.

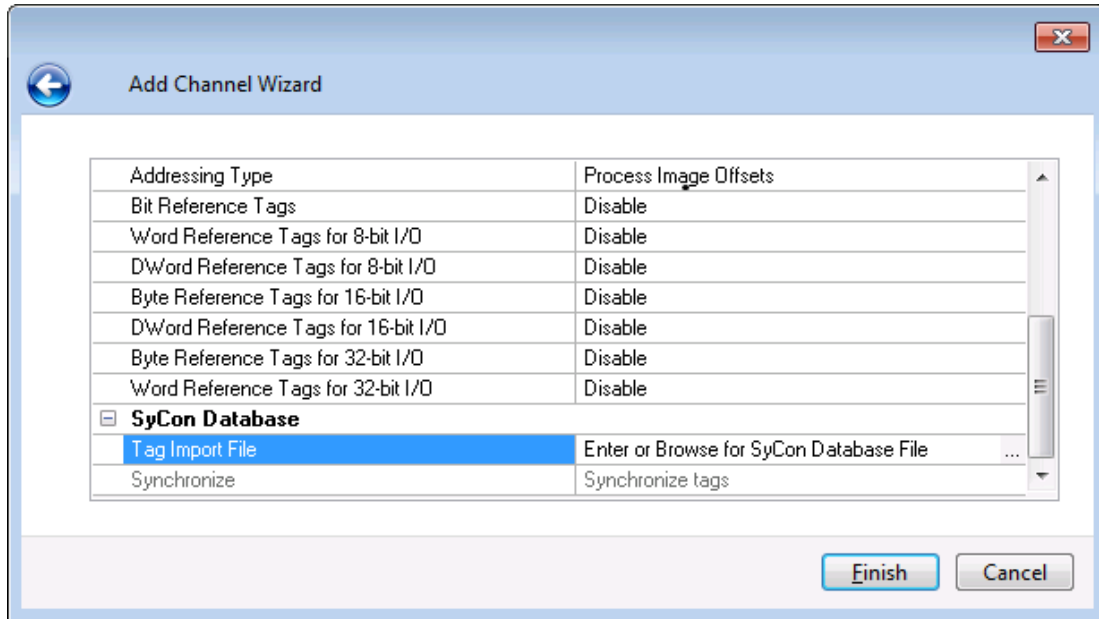




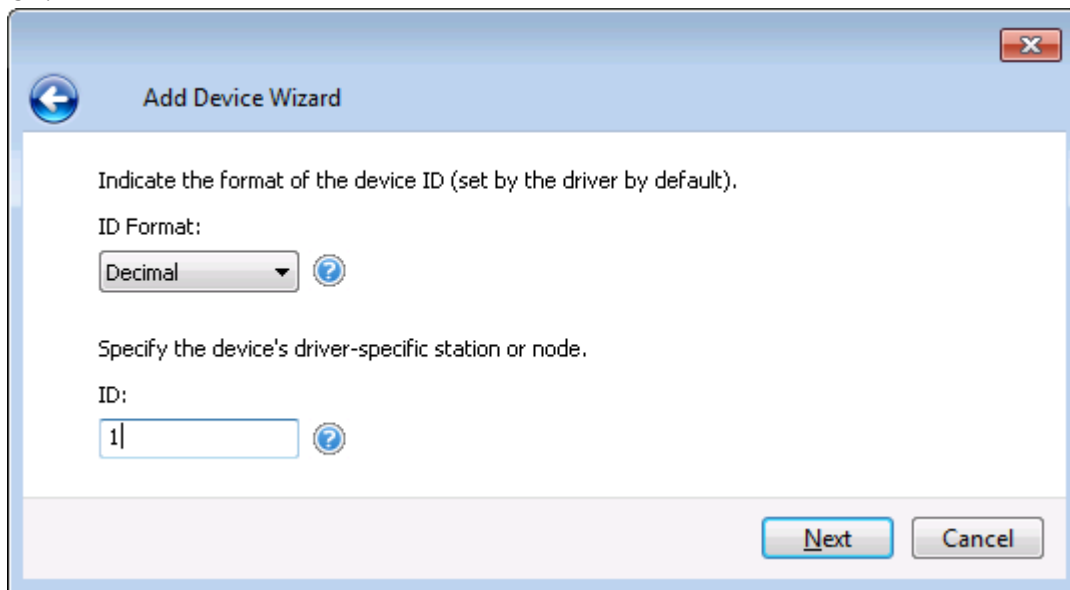
2. Set Expanded SyCon Tag Import to Enable.



3. In the New Channel Wizard summary, enter the Tag Imprt File by clicking on the ellipsis and browsing for the SyCon configuration file previously created.

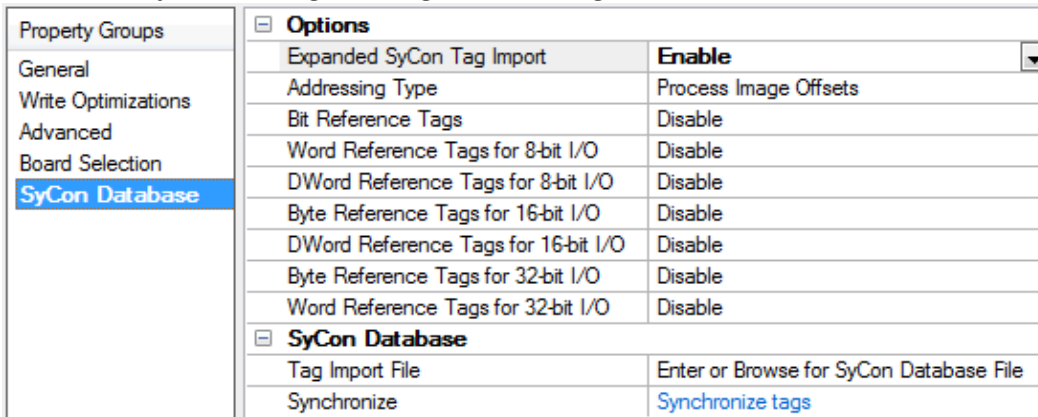


- Next, create a new device and set the Device ID to the MAC ID of the Slave board. In this example, it is 1.

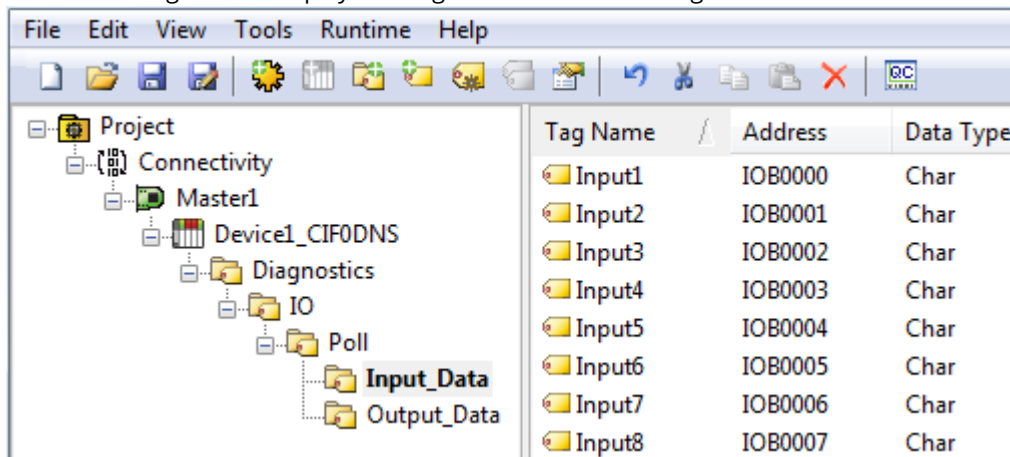


- Automatically generate the tags for the Slave. To do so, click **Channel Properties | SyCon Database**.

- Click on the “Synchronize tags” link to generate the tags.



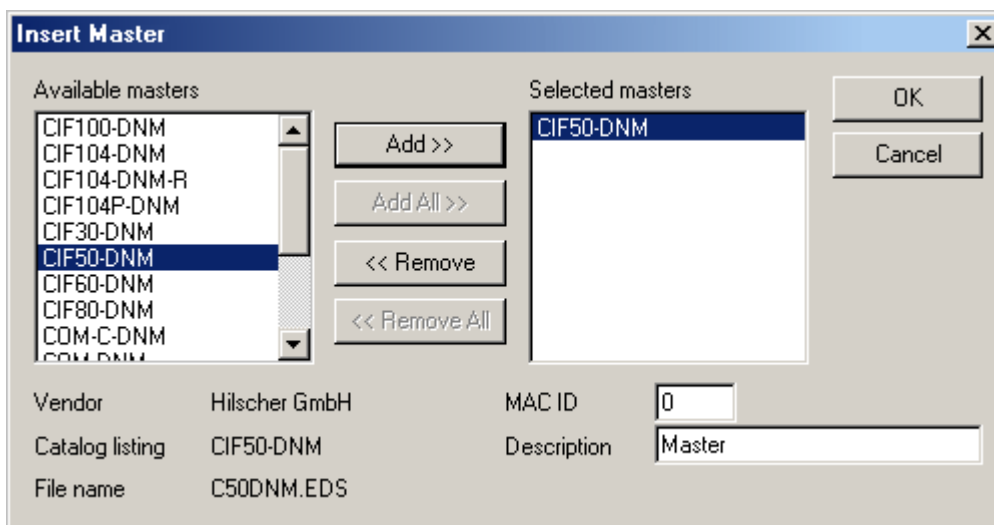
- Note:** The image below displays the tags created for the configured Slave board.



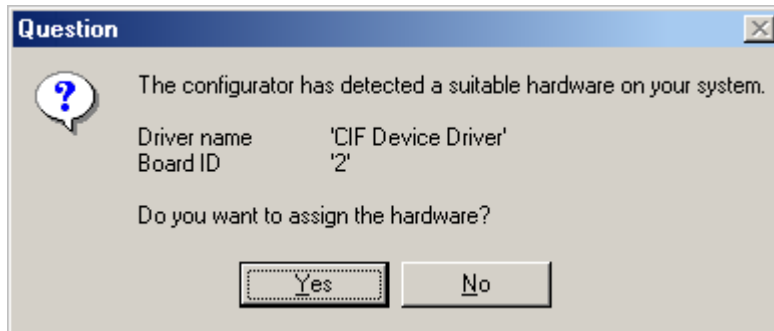
### PC 1 SyCon Configuration (Master)

Assume that the Slave board in PC 2 is configured for communications and access it from the DeviceNet Master in PC 1. A Master board configuration is required.

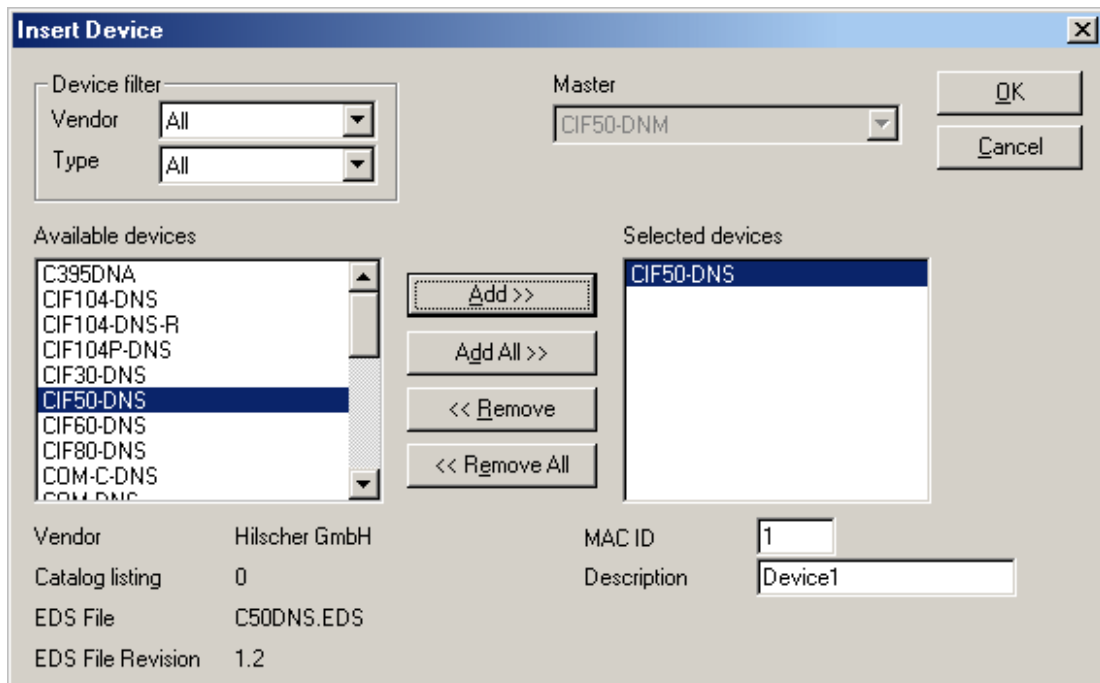
- First, open an empty SyCon project and insert the Master.



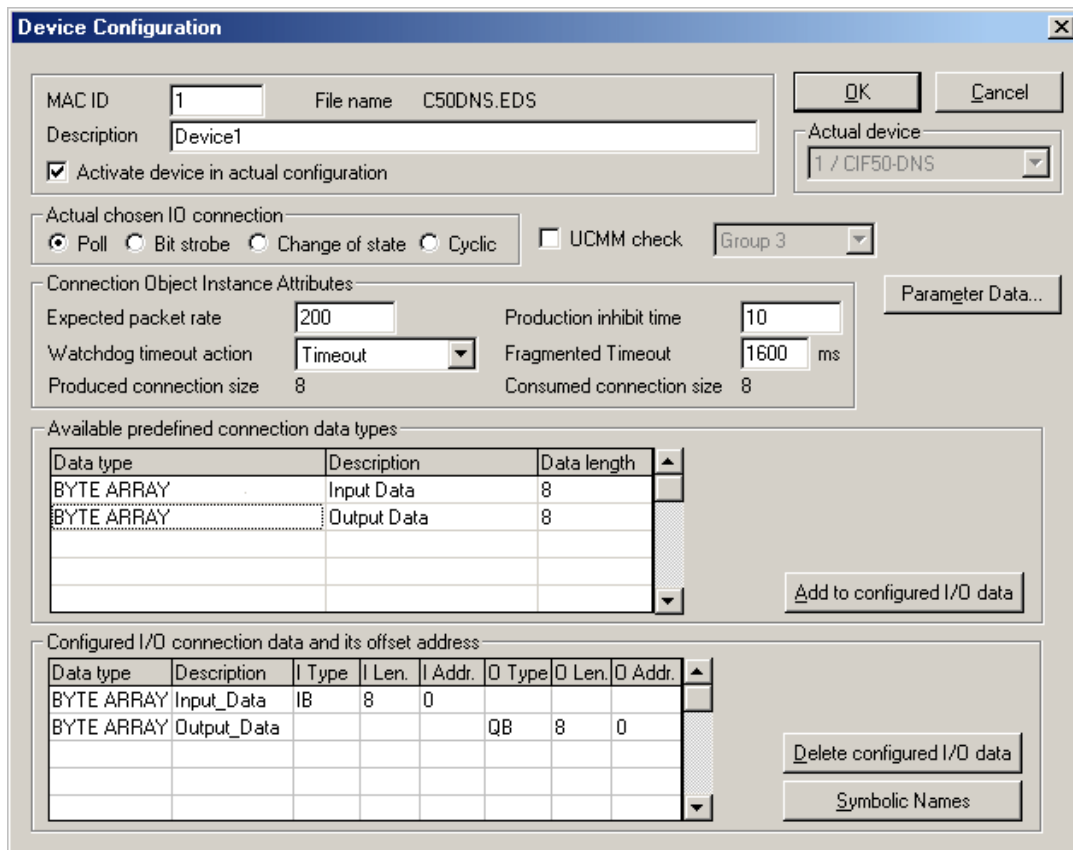
2. Assign it to the local Master board.



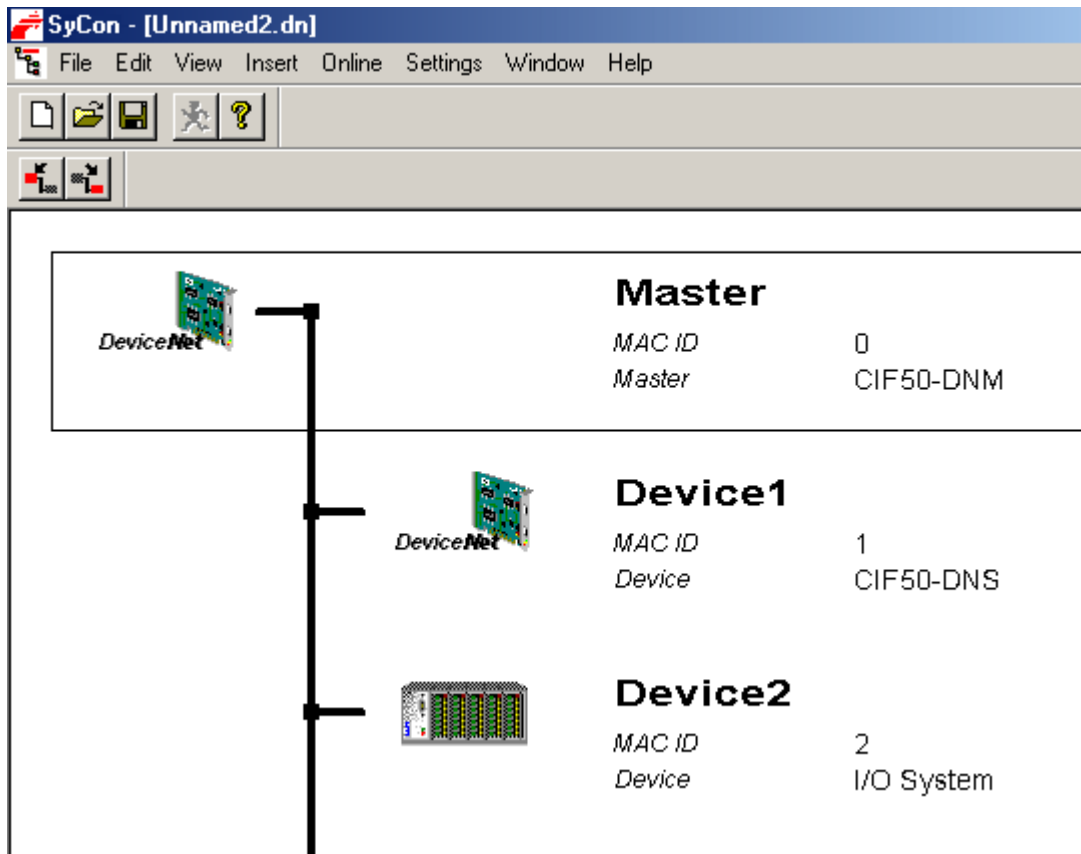
3. Next, insert the Slave. Alternatively, perform **Automatic Network Configuration**.
4. Assign the MAC ID chosen in Step 2 of PC 2 SyCon Configuration (Slave).



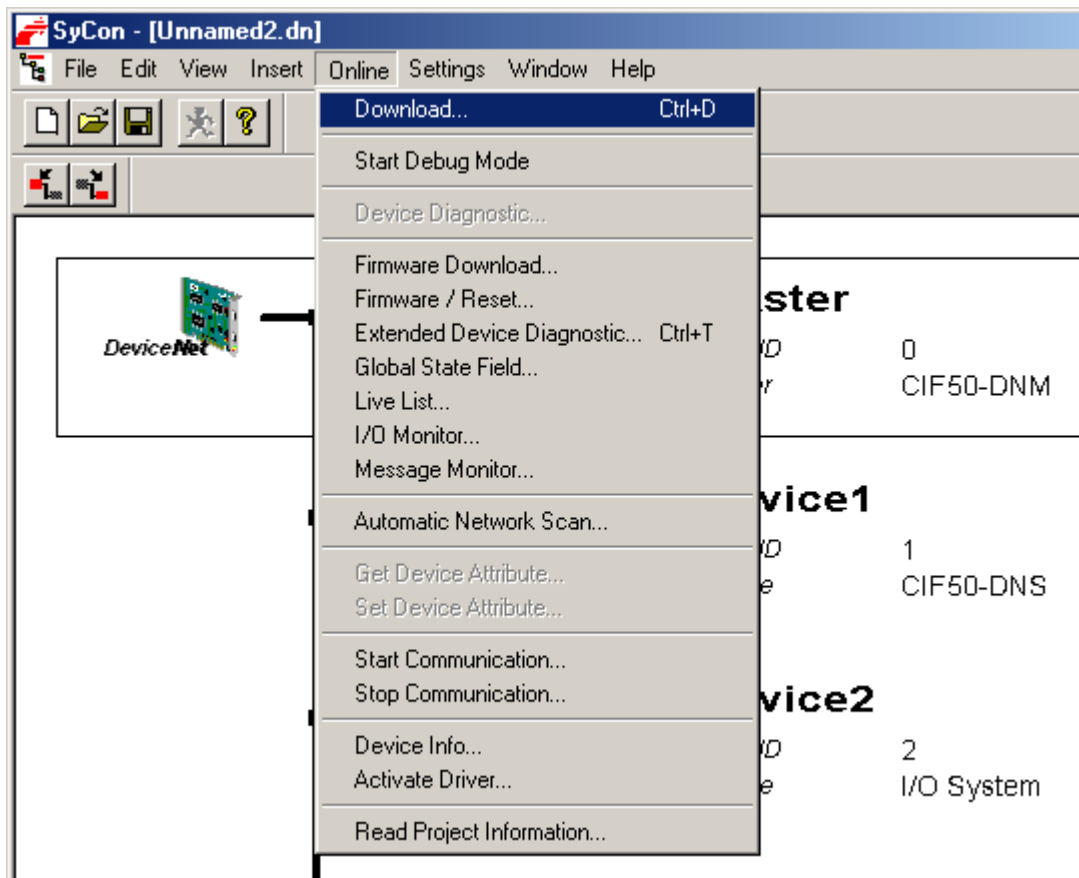
- **Note:** Do not assign the hardware when prompted if the Master board is in the same machine as the Slave board.
5. Configure the I/O. Its values must match the configured I/O in the Slave SyCon configuration created earlier.



6. If desired, insert the **Bar Code Reader Slave**. This is not required. The resulting network should appear as displayed below.



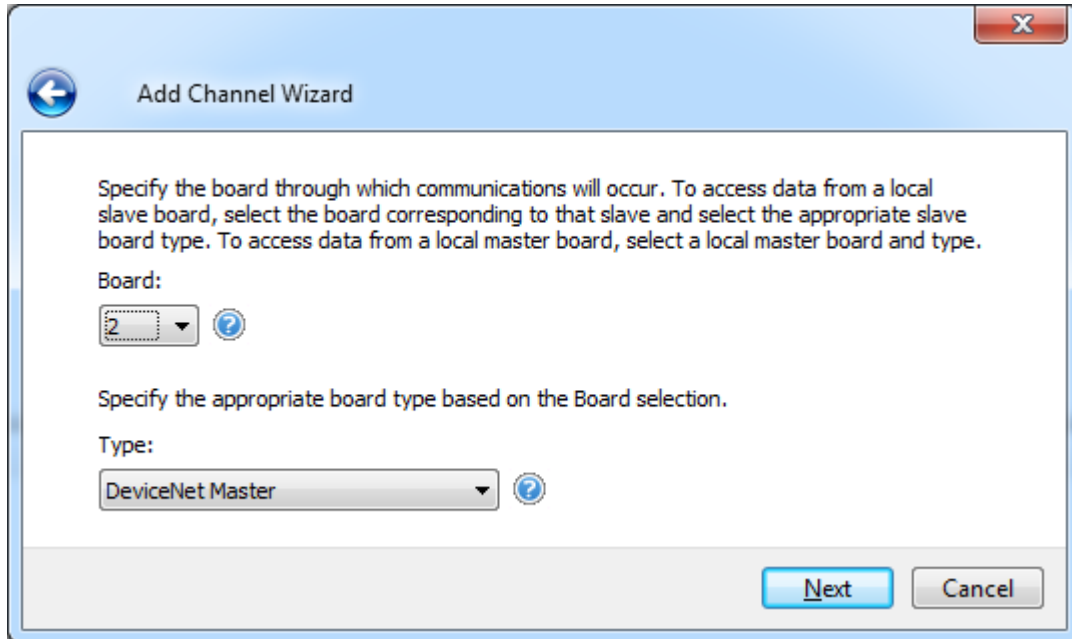
7. Save and then download the configuration to the Master board by clicking **Online | Download**.



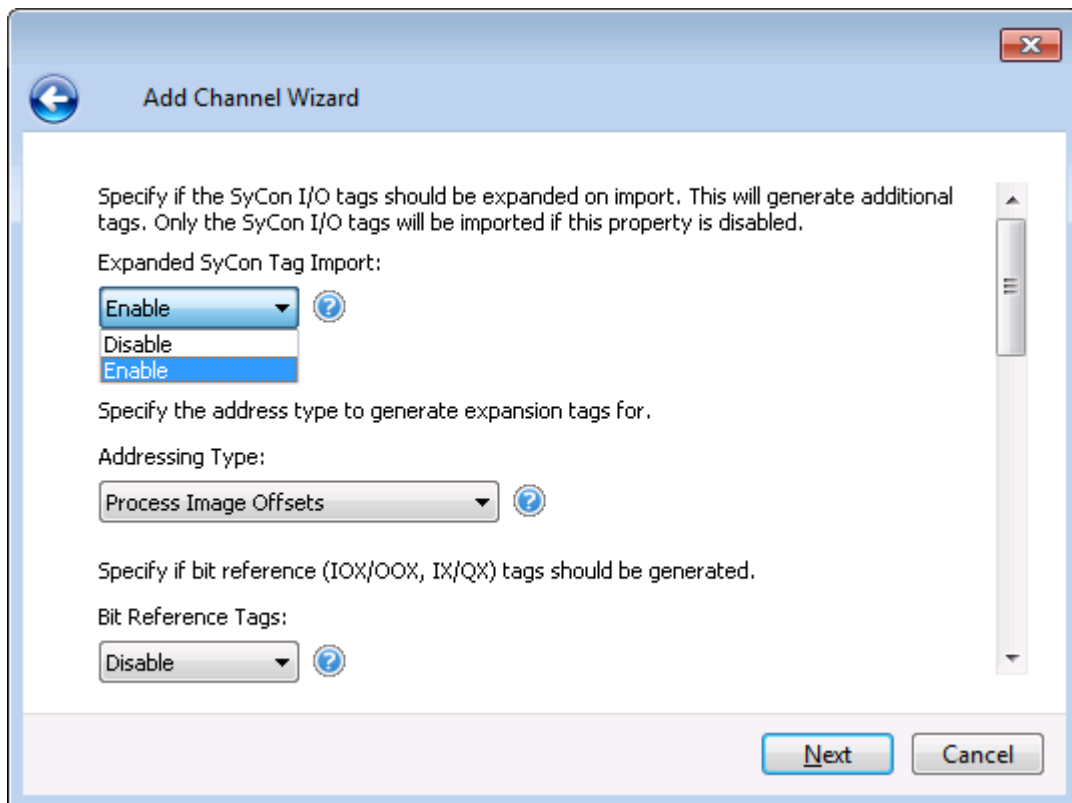
### PC 1 Server Configuration (Master)

Now that the Master board is configured, the Slave board can be accessed remotely using the Hilscher Universal Driver.

1. In the server, create a new channel. In the **Board Selection** dialog, choose the location of the board in the PC. For board type, select **DeviceNet Master**.

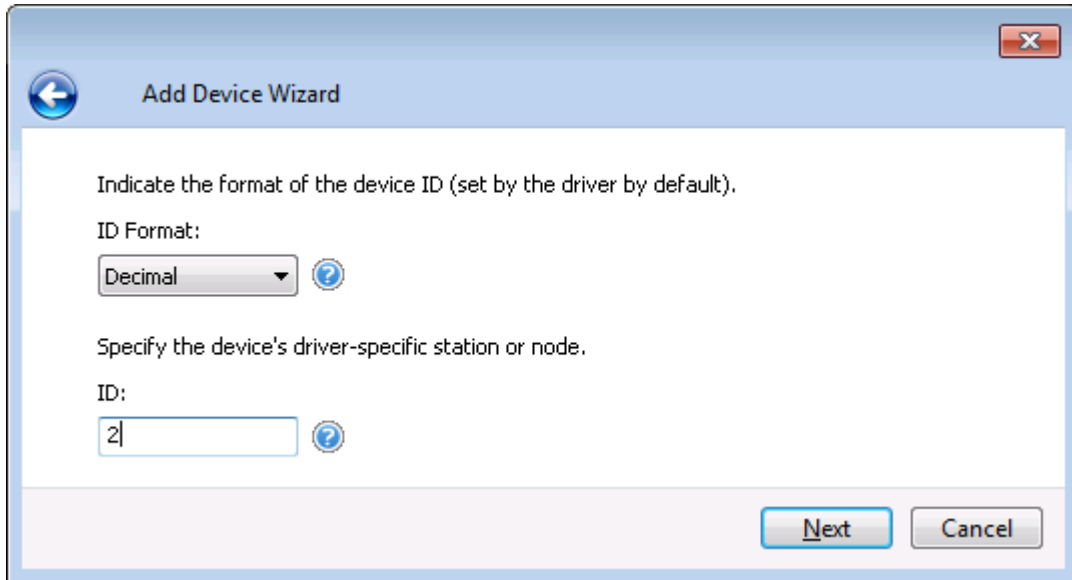


2. Set Expanded SyCon Tag Import to Enable.

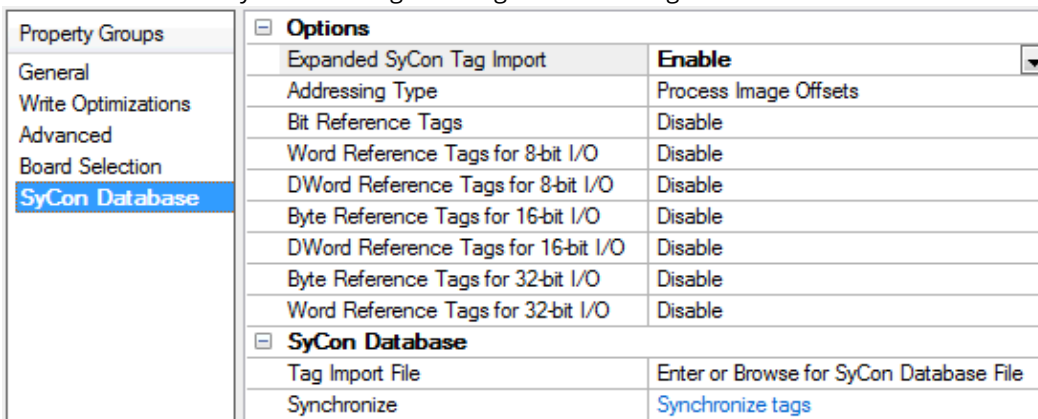


3. In the New Channel Wizard summary, summary, enter the Tag Import File by clicking on the ellipsis and browsing for the SyCon configuration file s previously created.
4. Next, create a new device and set the Device ID to the MAC ID of the Slave board in PC 2. In this example, it is 1. Then, create a new device and set the Device ID to the MAC ID of the Bar Code Reader. In this example, it is 2.

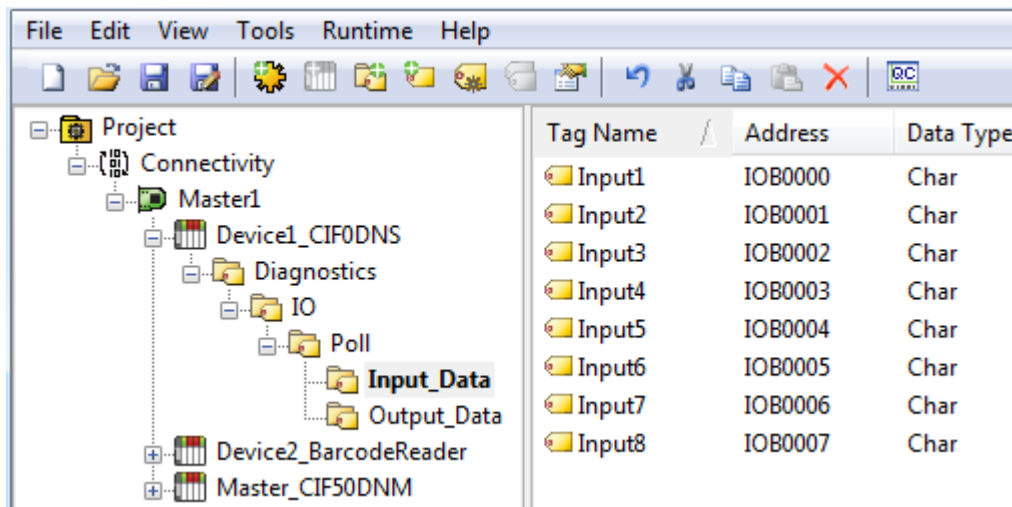





- Automatically generate the tags for the Slaves. To do so, click **Channel Properties | SyCon Database**. Click the Synchronize tags link to generate the tags.



- The image below displays the tags created for the Slave board in PC 2.



 **Important:** If the Slave board is in the same machine as the Master board, the Slave configuration may need to be redownloaded after downloading the Master configuration.

# Index

## 1

16-Bit Data Expansion 16

16-Bit Module Data 33, 37

## 3

32-Bit Data Expansion 17

32-Bit Module Data 33, 38

## 8

8-Bit Data Expansion 14

## A

Address '<address>' is out of range for the specified device or register 43

Address Descriptions 29

Addressing Type 11

Advanced Channel Properties 8

Allow Sub Groups 24

Array 29, 34

Array size is out of range for address '<address>' 44

Array support is not available for the specified address: '<address>' 44

Auto tag database generation cannot be performed while the driver is processing tags 51

Automatic Tag Database Generation 24

## B

BCD 28-29, 34

Big Endian 30, 35

Bit Reference Tags 12

Board Selection 8

Board Type for Board '<board number>' does not match the actual board installed. Verify Board Type and/or Board Selection 52

Board Type for Board '<board number>' does not match the Slave Type for one or more Slaves

configured. Delete or edit Slaves accordingly 52

Boolean 28-29, 34

Byte Addressing 32, 35

Byte Module 29, 34

Byte Swapping 30, 35

## C

Channel Assignment 21

Channel Properties - General 6

Channel Properties — Write Optimizations 7

Channel Setup 6

Char 29, 34

Communications Timeouts 26

Configured I/O 10

Connect Timeout 26

Connection Timeout 20

Create 24

## D

Data Collection 21

Data Type '<type>' is not valid for device address '<address>' 43

Data Types Description 28

dbm32.dll is not loaded and is required for auto tag generation. Verify SyCon is installed 52

Delete 23

Demote on Failure 27

Demotion Period 27

Description 21

Device '<device name>' is not responding 45

Device address '<address>' contains a syntax error 43

Device address '<address>' is Read Only 43

Device ID 20

Device Properties — Auto-Demotion 27

Device Properties — General 20

Device Properties — Tag Generation 22

Device Setup 20

Device Type 27

DeviceNet Master 9  
DeviceNet Slave 9  
DevOpenDriver () failed with error code '<code>' 45  
Diagnostic Tags 24  
Diagnostics 7  
Discard Requests when Demoted 27  
Do Not Scan, Demand Poll Only 22  
Driver 6, 21  
Duty Cycle 8  
DWord 28-29, 34  
DWord Module 18, 29, 34

## **E**

Error Codes 40  
Error Descriptions 39  
Expanded SyCon Tag Import 10, 25  
Expansion Settings 11  
Expansion Tags 10-11  
External Dependencies 5

## **F**

Float 28-29, 34

## **I**

I/O Data References 10  
I/O Data Tags 25  
IB/QB 16, 18  
ID 21  
ID/QD 15-16  
IEC Address Descriptions 34  
IEC Addressing 11  
IEEE-754 floating point 8  
Import AND Expand SyCon I/O Tags 10  
Information Imported From Database 24  
Information NOT Imported From Database 24

Initial Updates from Cache 22

Inter-Request Delay 26

IOB/OOB 16, 18

IOD/OOD 15-16

IOW/OOW 14, 18

IW/QW 14, 18

IX/QX 12

## **L**

LBCD 28-29, 34

Little Endian 30, 35

Long 28-29, 34

## **M**

Memory allocation error 45

Message Definitions 24

Missing address 43

Model 21

Module Format 24

## **N**

Name 20

Non-Normalized Float Handling 8

## **O**

On Duplicate Tag 23

Optimization Method 7

Overview 5

Overwrite 23

## **P**

Parent Group 23

PI 11

Process Image Address Descriptions 29  
Process Image Addressing 11  
Process Image Offset Addressing 11  
Profibus DP Master 9  
Profibus DP Slave 9

## **R**

Request All Data at Scan Rate 22  
Request Data No Faster than Scan Rate 22  
Request Timeout 20, 26  
Respect Client-Specified Scan Rate 22  
Respect Tag-Specified Scan Rate 22  
Retry Attempts 20, 26

## **S**

Scan Mode 22  
Short 28-29, 34  
Simulated 21  
Slave Board Configuration 53  
String 29, 34  
SyCon 6, 10, 29, 34, 53  
SyCon Configuration Database 24  
SyCon Database 9  
Symbolic Name 10  
Symbolic Names 24

## **T**

Tag Generation 22  
Tags Generated In Server 24  
The file is not a valid Sycon database or may be corrupt 51  
Timeouts to Demote 27  
Tutorial 53

## U

Unable to import from '<dll>' 44

Unable to load '<dll>' 44

Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics 49

Unable to read '<block size>' bytes starting at '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' 48

Unable to read '<block size>' device info bytes in area '<area>'. Board '<board>' returned Error Code '<code>' 46

Unable to read '<block size>' message bytes: msg.b <command>, msg.device\_adr <Device ID>... 49

Unable to read '<block size>' task state bytes in task '<task num>'. Board '<board>' returned Error Code '<code>' 47

Unable to read <block size> bytes starting at address <address> on device <device name> 46

Unable to read block size bytes starting from device. Board returned DNM Diagnostics 50

Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics 50

Unable to read tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics 49

Unable to read tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' 47

Unable to read tag '<name>': msg.b=<command>\_msg.device\_adr=<Device ID>... 48

Unable to read task state data in task '<task num>'. Board '<board>' returned Error Code '<code>' 47

Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DNM Diagnostics 51

Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned DPM Diagnostics 50

Unable to write to tag '<address>' from device '<device>'. Board '<board>' returned Error Code '<code>' 48

## W

Word 28-29

Word Addressing 30, 34

Word Module 17, 29, 34

Write All Values for All Tags 7

Write Only Latest Value for All Tags 7

Write Only Latest Value for Non-Boolean Tags 7

Write Optimizations 7