EtherCAT Slave Information

Annotation

Document: ETG.2001 G (R) V1.1.1

Nomenclature:

ETG-Number	ETG.2001
Type	G (Guide)
State	R (Release)
Version	V1.1.1

Created by: ETG
Contact: info@ethercat.org
Date: 2019-05-09

EtherCAT Technology Group

LEGAL NOTICE

Trademarks and Patents

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany. Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development. For that reason the documentation is not in every case checked for consistency with performance data, standards or other characteristics. In the event that it contains technical or editorial errors, we retain the right to make alterations at any time and without warning. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Copyright

DOCUMENT HISTORY

Version	Comment
1.0.0	Initial release Version
1.0.1	Clarified usage of ArrayInfo and of a Modules object dictionary Added new address of ETG Office Japan
1.0.2	Added attribute DependOnSlot for R/TxPdo:Index and R/TxPdo:Entry:Index for Example CiA 402 Drive Added PdoConfig and PdoAssign = True for Example CiA402 Drive and Example for Bus coupler
1.0.3	Correction in the element ModulePdoGroup@Alignment: The alignment of the coupler's Rx/TxPdos cannot be done by the ModulePdoGropu@Algignment. Correction in Table 8
1.0.4	Correction of ModulePdoGroup counting: First element of Slots:ModulePdoGroup aligns ModulePdoGroup 0, second element of Slots:ModulePdoGroup aligns ModulePdoGroup 1, etc. Correction in Figure 12 and Figure 14. Device:Type@ModulePdoGroup deleted, since it is not used.
1.0.5	CiA402 Module TxPdo SubIndex for simple data type object corrected to be 0 (rather than 1)
1.0.6	Mailbox:Coe:PdoConfig/PdoAssign/PdoUpload
1.0.7	Editorial, Typos in Mailbox:Coe:PdoConfig/PdoAssign/PdoUpload
1.0.8	Behaviour of PdoAssignment Object Download clarified when no Rx/TxPdo elements availablel in ESI fiel Accepting write access with default value to readonly entries not recommended
1.0.9	Update Enum data type description and picture
1.1.0	Addes clause SubDevice
1.1.1	Chapter "Objects with ENUM data type" revised.

CONTENTS

TABLES

FIGURES

1 Acknowledgement

The screenshots of the ESI shown are taken from ALTOVA™ XML Spy.

2 Glossary

SI subindex

3 Preface

This document describes the use of ESI elements and attributes on specific examples. It is an annotation to ETG.2000 EtherCAT Slave Information Specification.

4 Verification of Process Data Configuration

The PDO assignment (if ESI attribute *Mailbox:Coe@PdoAssign* = TRUE) and PDO configuration (if *Mailbox:CoE@PdoConfig* = TRUE) shall be downloaded from master to the slave while the slave is in PreOp state and before the master requests SafeOp state.

The slave shall accept the download of the PDO assignment (0x1C12 and 0x1C13) and PDO configuration (0x1Axx and 0x16xx). The slave only shall check the actual validity of the PDO assignment and configuration upon a state request to SafeOp. If the PDO assignment and/or configuration are invalid, the slave shall reject the state request to SafeOp, return to ErrPreOp and write the proper AL Status Code.

5 *Mailbox:Coe* attributes *PdoConfig*, *PdoAssign*, *PdoUpload*

5.1 Wording

5.1.1 PDO assignment

- List of PDO Mapping objects
- 0x1C12 describes the list of mapping objects in output direction (0x1600…0x17FF) and is related to the output Sync manager (in this case Sync manager 2)
- 0x1C13 describes the list of mapping objects in input direction (0x1A00…0x1BFF) and is related to the input Sync manager (in this case Sync manager 3)
- The following chapters use the term 0x1C12/0x1C13 for the PDO assignment objects; this may include further process data Sync manager assignment objects e.g. 0x1C14, 0x1C15, etc.
- The following chapters use the term 0x16nn/0x1Ann for output/input mapping objects

5.1.2 PDO configuration (= PDO mapping)

- One or more Lists of process data variables.
- Synonymously called PDO mapping.
- Objects 0x1600…0x17FF can be used to list the output process data variables.
- Objects 0x1A00…0x1BFF can be used to list the input process data variables.
- Process data variables are usually either in the in the range of 0x2000…0x5FFF (manufacturer specific) or 0x6000…0x9FFF (when a device profile such as CiA402 or ETG.5001 is used).
- In the following chapter, the term 0x16nn/0x1Ann is used for the PDO Mapping objects.

5.2 General

The flexibility of the process data configuration of a slave may different:

1. Fixed by design and not changeable.
2. Default configuration may be changeable/selectable or even completely freely configurable by the user using a configuration tool.
3. It cannot be configured offline and has to be read (=uploaded) from the slave.

In case (2) the possibly changed or defined configuration has to be written (downloaded) to the slave during start-up. To indicate this to the configtool the slaves ESI file attribute Mailbox:Coe@PdoAssign (and/or PdoConfig) shall be set TRUE.

In case (3) the configuration has to be uploaded from the configtool (master). To indicate this to the configtool the slave's ESI file attribute Mailobx:Coe@PdoUpload shall be set TRUE (PdoAssign and PdoConfig have to be FALSE in this case).

5.3 Use of PdoAssign

The ESI attribute *PdoAssign* is set to TRUE if 0x1C12/0x1C13 objects shall be downloaded during state change from PreOp to SafeOp.

If *PdoAssign* = TRUE the configtool shall generate all commands to write the PDO assignment objects.

If *Mailbox:CoE@CompleteAccess* = TRUE the PDO assignment objects shall be writeable via Complete Access.

Table 1 describes what an EtherCAT master/configtool and slaves have to do depending on *Mailbox:CoE@PdoAssign*.

Table 1: master/ configtool and slave behavior for PdoAssign

PdoAssign	**Master/configtool behaviour**		**Slave behaviour**	
	Mandatory	Optional	Mandatory	optional
TRUE	1. Generate 0x1C12/0x1C13 objects according to ESI elements Sm (NOTE: If Sm-element available the corresponding 0x1C12/0x1C13 shall be written even when no Rx/TxPdo elements available). The corresponding entries shall be generated according to the ESI elements RxPdo and TxPdo 2. Write Subindex0 of 0x1C12/13 = 0x00 3. Write subindex 1…n of 0x1C12/13 with 0x16nn/0x1Ann 4. Write SubIndex0 of 0x1C12/13 to highest subindex used.	• Step 2-4 may be done in one step by using complete access if it is supported by the slave. In this case writing SI0 to 0 is not required. • Support a work around if the slave rejects a PDO Assignment download	• Accept writing "0" for SI_0 in all objects from 0x1C12 to 0x1C1n • accept writing to SI1 to SIn according to the configured assignment.	A slave should confirm the state request from PreOp to SafeOp also when master does not write 0x1C12/0x1C13 and use the default configuration (as defined in the ESI file).
FALSE	0x1C12/0x1C13 is not downloaded to the slave		Slave does not need a download to 0x1C12/0x1C13 to confirm state request from PreOp to SafeOp.	Slave should accept when 0x1C12/0x1C13 are written with current configuration

5.3.1 Example

Two PDOs [2], 0x1A00 and 0x1A01, are described by the ESI file. One of them can be choosen (assigned) at a time. This assignment has to be downloaded to the slave during start-up (state change PreOp to SafeOp).

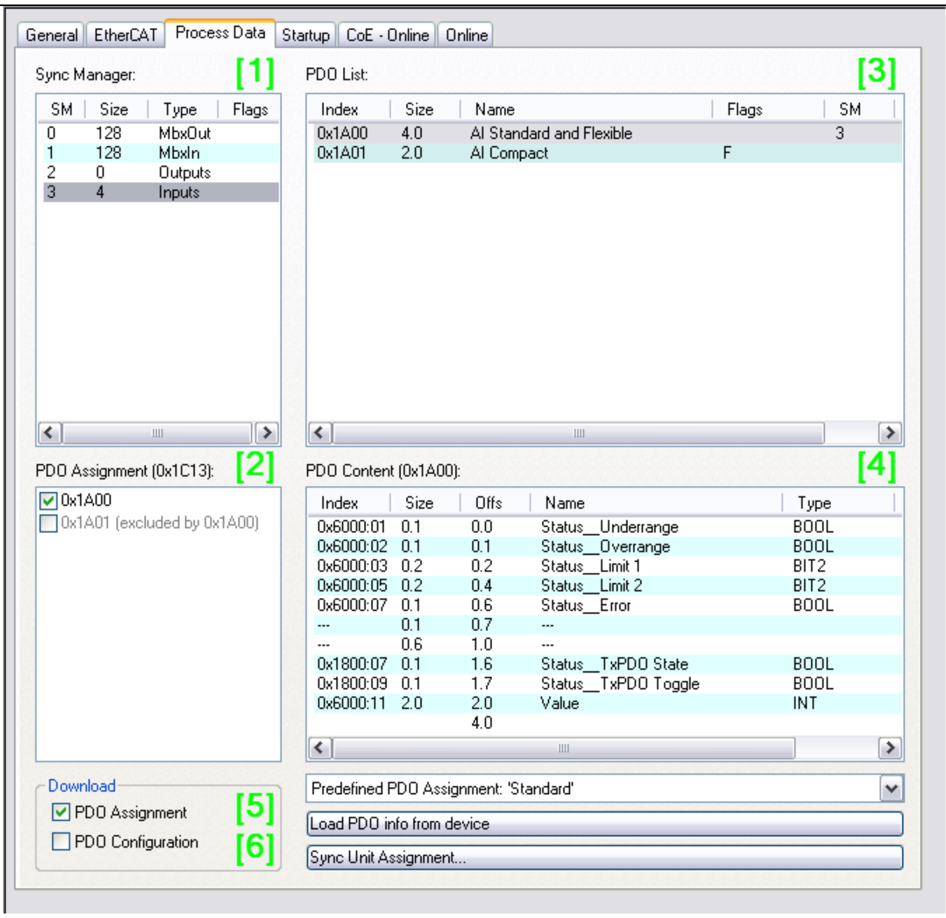
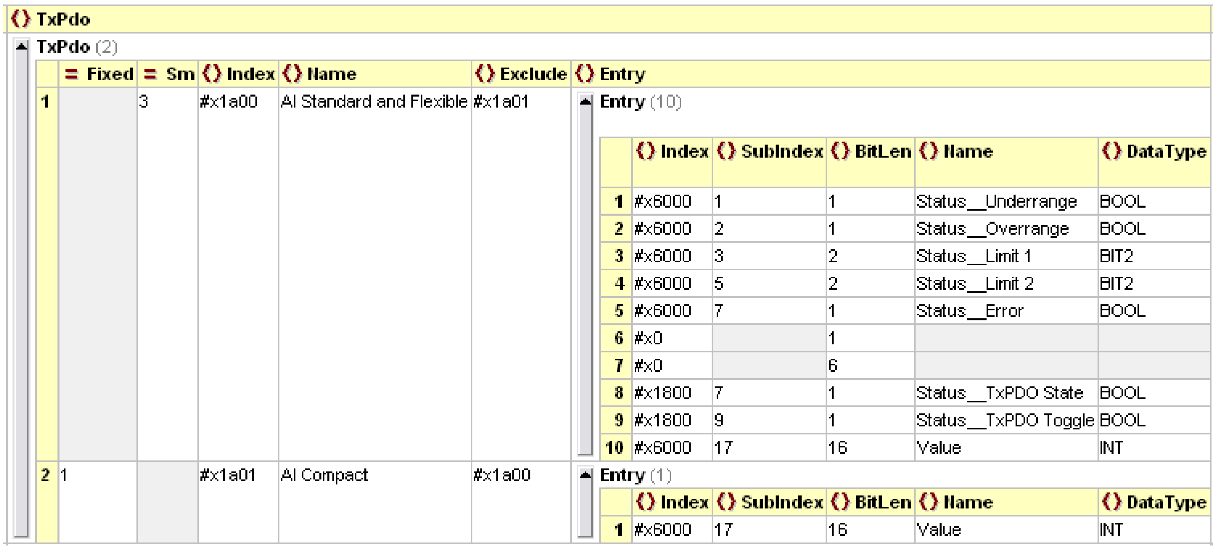


Figure 1: PdoAssign usage

In the ESI file, this PDO assignment looks as shown in Figure 3.



TxPdo

TxPdo (2)

	Fixed	Sm	Index	Name	Exclude	Entry
1		3	#x1a00	AI Standard and Flexible	#x1a01	Entry (10)
2	1		#x1a01	AI Compact	#x1a00	Entry (1)

Entry (10) of PDO 1:

	Index	SubIndex	BitLen	Name	DataType
1	#x6000	1	1	Status__Underrange	BOOL
2	#x6000	2	1	Status__Overrange	BOOL
3	#x6000	3	2	Status__Limit 1	BIT2
4	#x6000	5	2	Status__Limit 2	BIT2
5	#x6000	7	1	Status__Error	BOOL
6	#x0		1		
7	#x0		6		
8	#x1800	7	1	Status__TxPDO State	BOOL
9	#x1800	9	1	Status__TxPDO Toggle	BOOL
10	#x6000	17	16	Value	INT

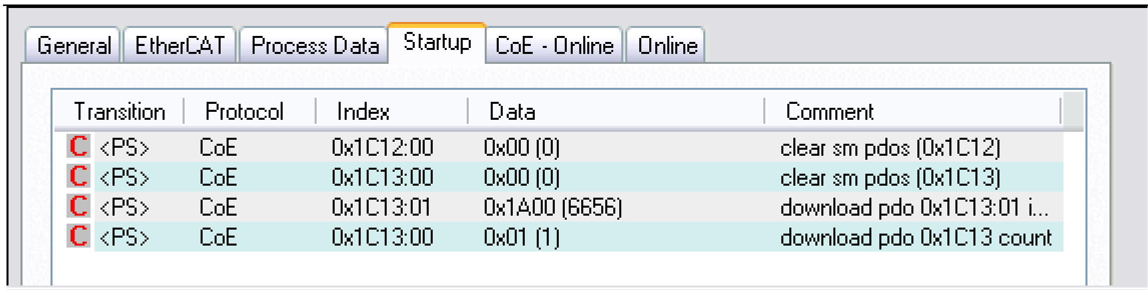
Entry (1) of PDO 2:

	Index	SubIndex	BitLen	Name	DataType
1	#x6000	17	16	Value	INT

Figure 2: PdoAssign description in the ESI file

By using the attribute *TxPdo@Sm* the PDOs can be assigned to a Sync manager by default. In this case the PDO 0x1A00 is assigned to Sync manager 3, while 0x1A01 is not assigned by default.

If PDO 0x1A00 is assigned the following PDO assignment commands (SDO commands) have to be generated by the configtool:



Transition	Protocol	Index	Data	Comment
C <PS>	CoE	0x1C12:00	0x00 (0)	clear sm pdos (0x1C12)
C <PS>	CoE	0x1C13:00	0x00 (0)	clear sm pdos (0x1C13)
C <PS>	CoE	0x1C13:01	0x1A00 (6656)	download pdo 0x1C13:01 i...
C <PS>	CoE	0x1C13:00	0x01 (1)	download pdo 0x1C13 count

Figure 3: PDO Assignment downloaded with start-up commands

First subindex 0 is written to “0” to indicate to the slave that the PDO assignment is changed. Then the PDO assignment of the input PDO 0x1C13 is changed. In this case, the PDO assignment list has only one entry in subindex 01, which is PDO 0x1A00.

After this, the PDO assignment is activated by writing 0x1C13:01 to the highest subindex in this object, which is “1” in this case.

The PDO assignment start-up commands shall be downloaded in in the top-down order shown in Figure 3.

5.4 PdoConfig

The ESI attribute *PdoConfig* is set = TRUE if the PDO configuration 0x16nn/0x1Ann shall be downloaded during state change from PreOp to SafeOp.

For a slave design, it is better to avoid the use of *PdoConfig* since it makes the slave code simpler and smaller and the start-up of the network faster.

RECOMMENDATION: PDO Configuration Objects should always be writable with their actual values, also when they are readonly.

If *PdoConfig* = TRUE the configtool shall generate all commands to write the PDO configuration with *Pdo@fixed* = FALSE.

RECOMMENDATION: PDOs which have the attribute *Pdo@fixed* = true shall not be written.

If *CompleteAccess* = TRUE the PDO configuration objects shall be writable by Complete Access.

Table 2 describes what an EtherCAT master/configtool and slaves have to do depending on *Mailbox:Coe@PdoConfig*.

Table 2: master/ configtool and slave behavior for PdoConfig

PdoConfig	**Master/configtool behaviour**		**Slave behaviour**	
	Mandatory	Optional	Mandatory	optional
TRUE	1. Generate 0x16nn/0x1Ann objects according to ESI elements TxPdo and RxPdo or to the offline configuration done by user 2. If *PdoAssign*=1: Write SI0 of 0x1C12/0x1C13 with 0x00 3. Write SI0 of 0x16nn/0x1Ann with 0x00 4. If *PdoAssign*=1: write SI1…n of 0x1C12/13 with 0x16nn/0x1Ann 5. write SI1…n of 0x16nn/0x1Ann with index of variables (e.g. 0x2nnn, 0x6nn, 0x7nnn) 6. Write SI0 of 0x16nn/0x1Ann to highest subindex used If PdoAssign = 1 7. write SI0 of 0x1C12/13 to highest subindex used NOTE: If *Pdo@Fixed* = 1 the configtool may generate the commands to write 0x16nn/0x1Ann. The master has to handle a possible abort code for objects with *Pdo@fixed*.	If Complete access is supported the PDO mapping object may be written in one go (without set SI0 to 0.	PDOs shall always be writable in PreOp state, independent of whether the Pdo is used or not	A slave should confirm the state request from PreOp to SafeOp also when master does not write 0x16nn/0x1Ann and use the default configuration (as defined in the ESI file)

PdoConfig	Master/configtool behaviour		Slave behaviour	
FALSE	Configtool does not generate and commands to write 0x16nn/0x1Ann (even then when Pdo@Fixed = 0)	----	Slave does not need a download to 0x16nn/0x1Ann to confirm state request from PreOp to SafeOp.	Slave should accept when 0x16nn/0x1Ann are written with current configuration

5.4.1 Example

The list of variables [4] of the Input PDO 0x1A00 can be changed. This can also be seen in [3] as there is no “Fixed” (no “F” in column Flags) flag with this PDO.

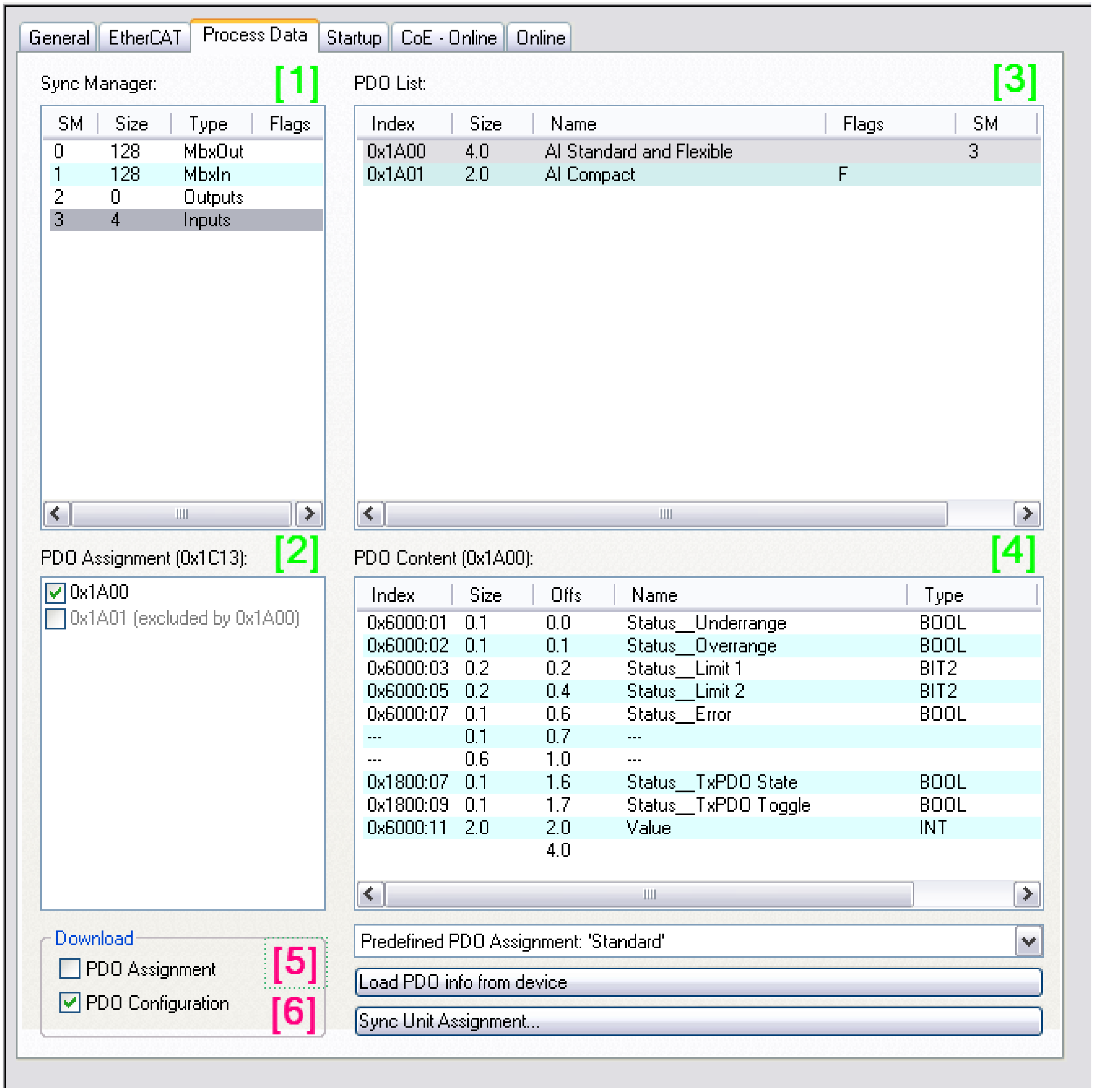
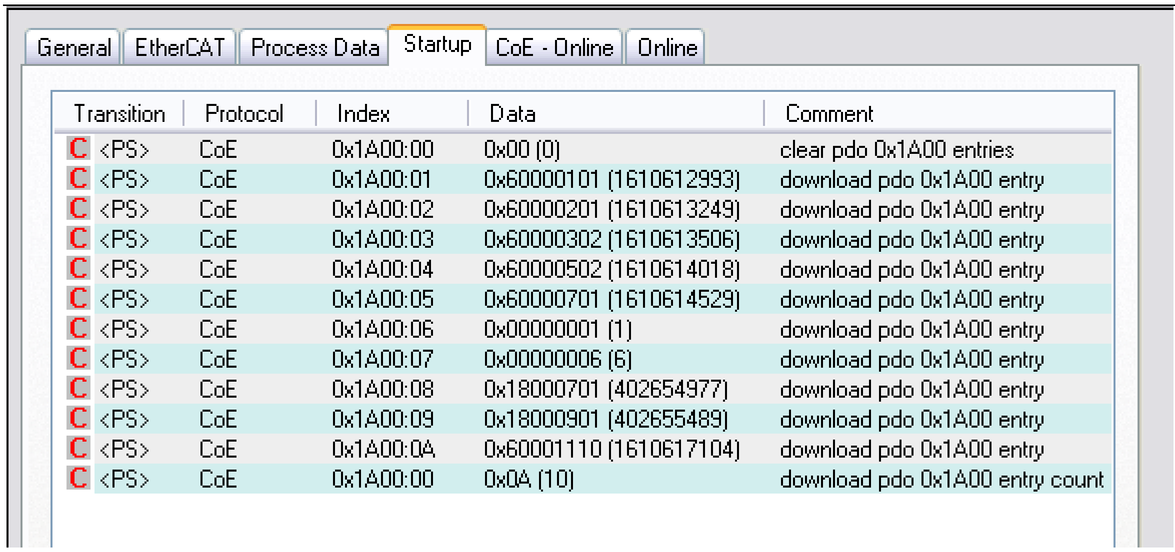


Figure 4: PdoConfig usage

The ESI element *Mailbox:Coe@PdoConfig* is set to TRUE [6]. For the configtool, this means to generate the PDO configuration start-up commands as shown in Figure 5.



Transition	Protocol	Index	Data	Comment
C <PS>	CoE	0x1A00:00	0x00 (0)	clear pdo 0x1A00 entries
C <PS>	CoE	0x1A00:01	0x60000101 (1610612993)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:02	0x60000201 (1610613249)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:03	0x60000302 (1610613506)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:04	0x60000502 (1610614018)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:05	0x60000701 (1610614529)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:06	0x00000001 (1)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:07	0x00000006 (6)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:08	0x18000701 (402654977)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:09	0x18000901 (402655489)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:0A	0x60001110 (1610617104)	download pdo 0x1A00 entry
C <PS>	CoE	0x1A00:00	0x0A (10)	download pdo 0x1A00 entry count

Figure 5: PDO Configuration downloaded with start-up commands

Figure 5 describes the SDO commands for the PDO configuration how they need to be downloaded (top to bottom).

To indicate that the PDO configuration of 0x1A00 will be changed now 0x1A00:00 is set to “0”. Then, subindex by subindex is written with the index and subindex of the variables, e.g.

- 0x1A00:01 is written with index = 0x6000 | subindex = 01 | bit length = 01
- 0x1A00:02 is written with index = 0x6000 | subindex = 02 | bit length = 01
- 0x1A00:03 is written with index = 0x6000 | subindex = 03 | bit length = 02
- 0x1A00:06 is written with index = 0x0000 | subindex = 00 | bit length = 01 (padding bit)
- …..

After writing the PDO configuration 0x1A00:00 is set to the value of the highes subindex 0x0A to indicate that the PDO configuration is now valid.

5.5 PdoUpload

Table 3 describes what an EtherCAT master/configtool and slaves have to do, depending on *Mailbox:Coe@PdoUpload.*

Table 3: master/ configtool and slave behavior for PdoUpload

PdoUpload	**Master/configtool behaviour**		**Slave behaviour**	
	Mandatory	Optional	Mandatory	Optional
TRUE	1. Upload 0x1C12/0x1C13 2. Upload 0x16nn/0x1Ann listed in 0x1C12/0x1C13 3. Upload PDO entries (e.g. 0x2nnn, 0x6nnn, 0x7nnn) listed in 0x1Ann/0x16nn	If SdoInfo = TRUE: upload entry description of PDO entries to get name, data type	0x1C12, 0x1C13 0x16nn, 0x1Ann PDO entries shall be accessible	if SdoCompleteAccess = TRUE: PDO objects shall be accessible by Complete Access
FALSE	---	----	0x1C12, 0x1C13 0x16nn, 0x1AnnPDO entries shall be accessible	if SdoCompleteAccess = TRUE: PDO objects shall be accessible by Complete Access

6 DC /OpModes

6.1 DC with Input Latch (Dc:OpMode:ShiftTimeSyncX@Input)

Table 4: Used ESI elements and attributes

Element Name
Dc:OpMode:ShiftTimeSyncX
Dc:OpMode:ShiftTimeSyncX@Input

Usually the sync event of the slave is used to trigger the local output event. However, for input slaves it is more useful to trigger the input event with the SyncSignal.

In Figure 6, A, the Sync0 signal of the slave is shifted by *ShiftTimeSync0* (*Input* = FALSE) from the DC Base signal in positive direction so that the Sync0 signal and, hence, the input latch is triggered late so that the input data are quite new before the EtherCAT frame collects them for the master.

The same result is reached when the *ShiftTimeSync0* is calculated in negative direction from the DC Base Signal (*Input* = TRUE).

To achieve the same result for Input = FALSE when the master cycle time is changed the ShiftTimeSync0 has to be changed, too, so that the input values are latched as late as possible.

However, if *Input* = TRUE, the master calculates the Sync0Signal again in negative direction from the DC Base Signal and achieves the same result without any changes.

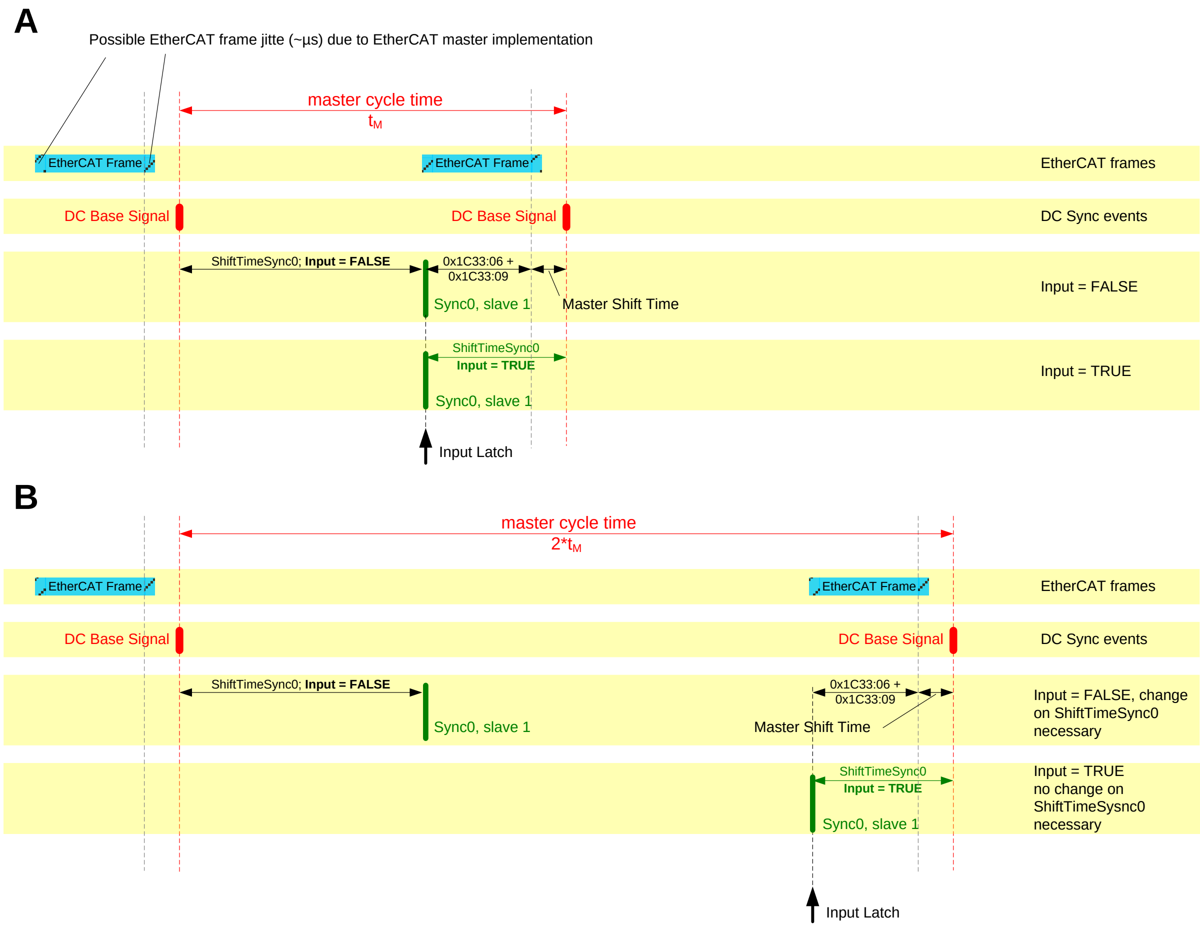


Figure 6: Settings for late Input Latch

7 Modules/Slots

In the following clause several examples for the use of the *Modules* and *Slots* elements are shown. The elements Modules and Slots are used to describe devices which are based on ETG.5001 Modular Device Profile.

The profile is used for devices which have a flexible physical and/or logical structure. Physical modules or logical functionality are all describes as modules. A module has its own process data and start-up commands which are described in the *Modules* element.

The allowed order and combinations of those possible modules are described by the Slots element. Exactly one *Module* can be connected to one slot.

Indexes of the mapping PDOs (indexes start at 0x1600 and 0x1A00) and of the PDO entries (index range start at 0x6000 – 0x9FFF) may be adapted with every slot and/or with every slot group.

The description of the module's object dictionary can be done in the element Module:Profile:Dictionary. To adopte the object index to the slot number the attributes Module:Profile:Dictionary:Objects:Object:Index@DependOnSlot or Module:Profile:Dictionary:Objects:Object:Index@DependOnSlotGroup may be used.

7.1 Example CiA402 Drive

Reference File with example: ETG2001_ModulesSlots_CiA402.xml

This example describes a multi-axis-drive according to CiA402 making the following assumptions. All possible combinations of Function Groups/ Operation Modes are described as one module each.

- Device supports up to 2 axis
- Both axis support different Functions Groups and operation modes in the following combinations.
    - Synchronous with Sync manager event (= process data) is default:
        - Position Mode
        - Position Mode with Touch Probe and Homing Mode
        - Velocity Mode
        - Velocity mode with Touch Probe and Homing Mode
    - Synchronous with Distributed Clocks is default:
        - Position Mode
        - Position Mode with Touch Probe and Homing Mode
        - Velocity Mode
        - Velocity mode with Touch Probe and Homing Mode
- The CiA402 Drive Profile describes an index area for each axis of 0x800 indexes. The indexes of the process data values such as control word or position demand value correspond to IEC61800-7-201 / ETG.6010.

7.1.1 Used ESI elements

The elements and attributes listed by Table 5 are used in the following example.

Table 5: Used ESI elements and attributes

Element Name
Module:Type
Module:Type:ModuleIdent
Module:Name
Module:RxPdo
Module:TxPdo
Module:RxPdo:Entry:Index@DependOnSlot
Module:TxPdo:Entry:Index@DependOnSlot
Module:Profile
Module:DcOpModeName
Slots@SlotPdoIncrement
Slots:Slots@SlotIndexIncremenet
Slots:Slot@MinInstances
Slots:Slot@MaxInstances
Slots:Slot:Name
Slots:Slot@SlotGroup
Slots:Slot:ModuleIdent
Slots:Slot:ModuleIdent@Default

7.1.2 Description of Modules

The combinations of Function Groups and operation modes defined in 7.1 are transferred to the description within the *Modules* element. The first thing is do assign an identifier (*Type@ModuleIdent*) to each module which is vendor specific (choose freely). The text of Type gives the name of this Module while *Name* gives a more detailed description of this module.

Modules
Module (8)

	Type			Name	RxPdo	TxPdo	Profile	DcOpModeName
1	Type	ModuleIdent	#x0100	Position Mode; synchronous with process data	RxPdo (1)	TxPdo (1)	Profile	Synchron
		Text	Position Mode					
2	Type	ModuleIdent	#x0101	Position Mode; synchronous with Distributed Clocks	RxPdo (1)	TxPdo (1)	Profile	DC
		Text	Position Mode (DC)					
3	Type	ModuleIdent	#x0110	Position Mode including Homing and Touch Probe Functionality; synchronous with process data	RxPdo (3)	TxPdo (3)	Profile	Synchron
		Text	Position Mode					
4	Type	ModuleIdent	#x0111	Position Mode including Homing and Touch Probe Functionality; synchronous with Distributed Clocks	RxPdo (3)	TxPdo (3)	Profile	DC
		Text	Position Mode | Homing | Touch Probe (DC)					
5	Type	ModuleIdent	#x0200	Velocity Mode; synchronous with process data	RxPdo (1)	TxPdo (1)	Profile	Synchron
		Text	Velocity Mode					
6	Type	ModuleIdent	#x0201	Velocity Mode; synchronous with process data	RxPdo (1)	TxPdo (1)	Profile	DC
		Text	Velocity Mode (DC)					
7	Type	ModuleIdent	#x0210	Position Mode including Homing and Touch Probe Functionality; synchronous with Distributed Clocks	RxPdo (3)	TxPdo (3)	Profile	Synchron
		Text	Velocity Mode | Homing | Touch Probe					
8	Type	ModuleIdent	#x0211	Velocity Mode including Homing and Touch Probe Functionality; synchronous with Distributed Clocks	RxPdo (3)	TxPdo (3)	Profile	DC
		Text	Velocity Mode | Homing | Touch Probe (DC)					

Figure 7: DS402 Example – element “Modules“

The default synchronization mode in which the axis is configured when one of the modules is configured is named by the DcOpModeName. It references to the *Device:Dc:OpMode:Name* element.

The process data of each module is described by the *RxPdo* and *TxPdo* elements. The PDO indexes and PDO entry indexes are described as if they were only for the first module. The adaption of the indexes for the following axis is defined within the *Slots* element.

The Sync manager assignment is related to the Sync managers of the device as described in the element *Device:Sm*.

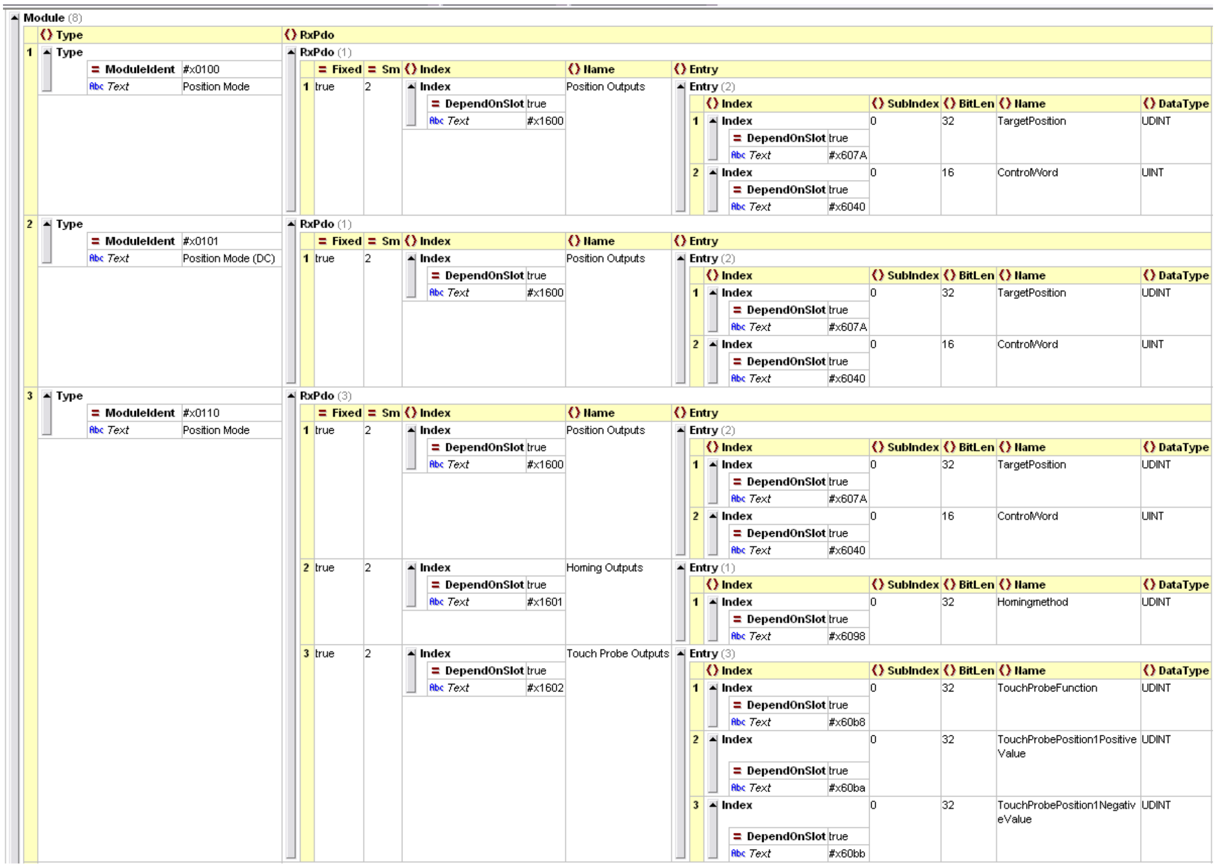


Figure 8: RxPdo description of modules

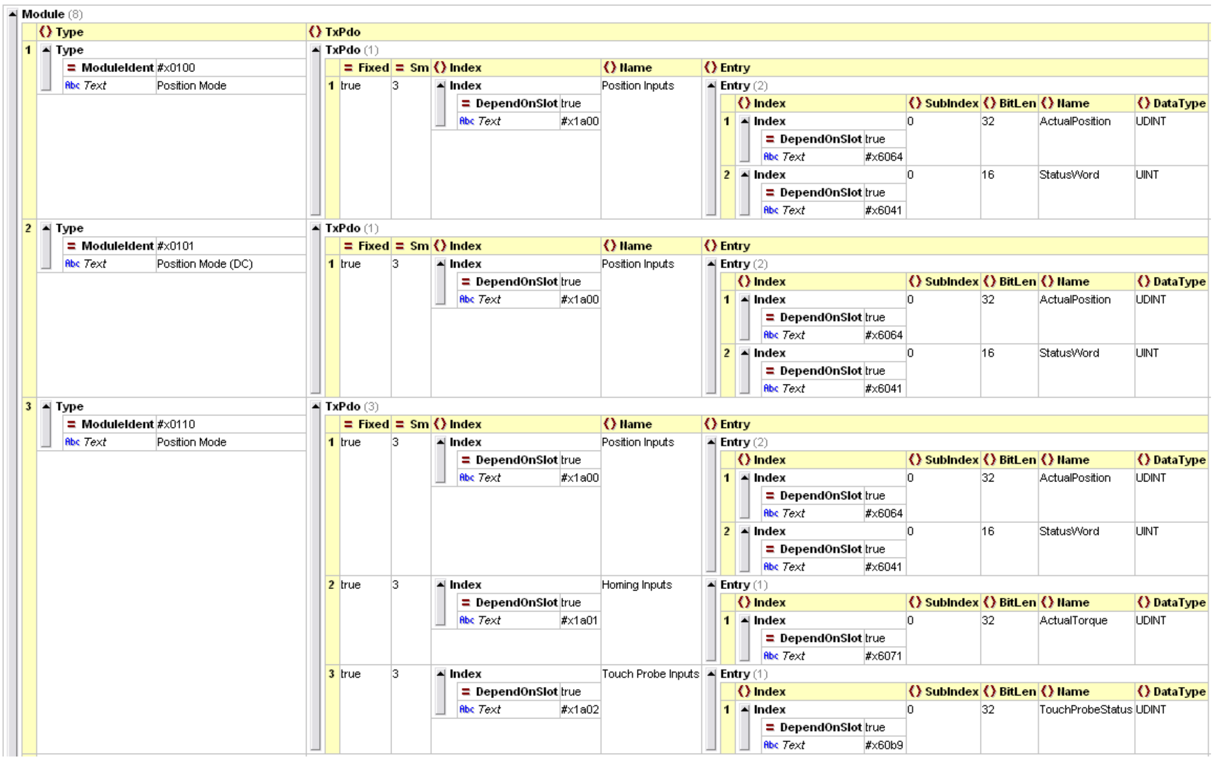


Figure 9: TxPdo description of modules

The attribute *Profile@ProfileNo* states the profile according to which the module is defined. EtherCAT configuration tools with interface to a motion controller allow generating the corresponding CiA402 axis in the motion controller if *ProfileNo* is 402.

7.1.3 Description of Slots

The two axes are represented by one slot each and are named “Axis 1” and “Axis 2”. Which mode is configured for the axis is decided by the assigned module. Exactly one *Mode* can be configured for each axis, which is defined by *MinInstances* = *MaxInstances* = 1.

All modes (=*Modules*) which can be configured for one axis (=*Slot*) are listed in the *ModuleIdent* element. The *Module*, resp. default mode configured for both axis is the one with *ModuleIdent* = 100 (= Position Mode). Of course, this can be replaced by the other ones listed.

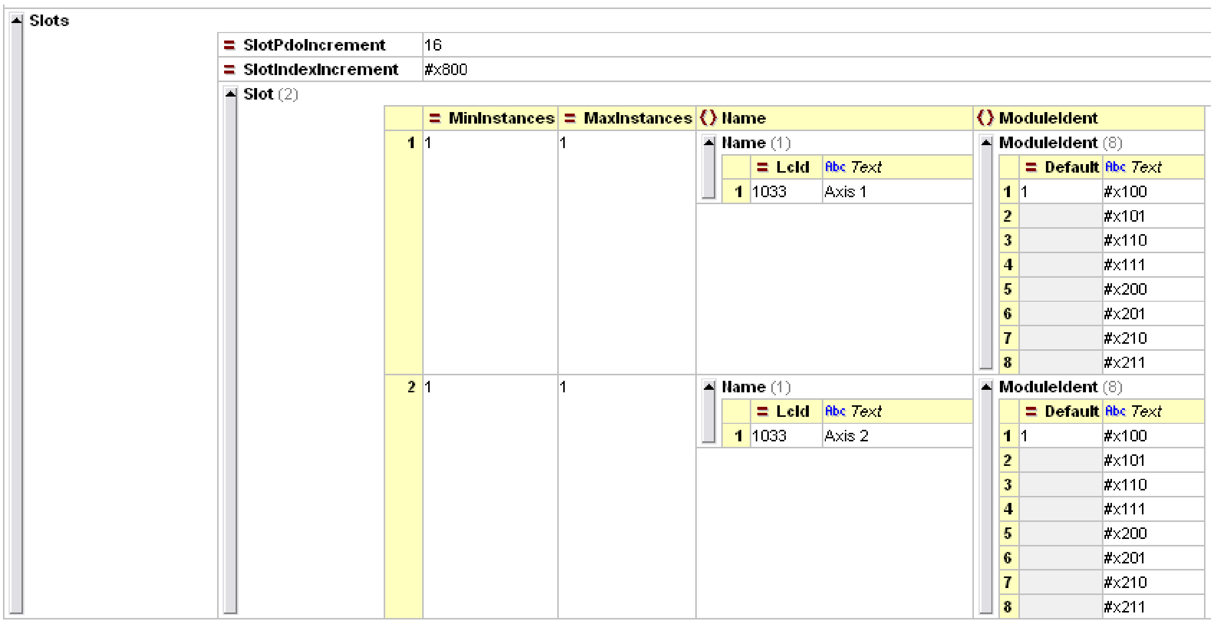


Figure 10: DS402 example – element Slots

The index increment for the PDOs and PDO entries of the two axes is defined by the SlotPdoIncrement and SlotIndexIncrement as described by Table 6. That they become effective for the PDOs and PDO entry object indexes the attributes *Module:R/TxPdo:Entry:Index@DependOnSlot* and *Module:R/TxPdo:Index@DependOnSlot* have to be TRUE.

Table 6: PDO Index and PDO Entry Index Increment

Axis No	RxPDO	RxPDO Entries		TxPDO	TxPDO Entries	
1	0x16**00**	0x6**0**7A	0x6**0**40	0x1A**00**	0x6**0**64	0x6**0**41
	↓	↓	↓	↓	↓	↓
	SlotPdoIncrement = 16 (0x10)	SlotIndexIncrement = 0x800		SlotPdoIncrement = 16 (0x10)	SlotIndexIncrement = 0x800	
	↓	↓	↓	↓	↓	↓
2	0x16**10**	0x6**8**7A	0x6**8**40	0x1A**10**	0x6**8**64	0x6**8**41

The PDO configuration and PDO assignment is automatically generated by the configuration tool depending on the users modules selection. This configuration has to be downloaded to the slave which is done when the attributes *Mailbox:Coe@Assign* and *Mailbox:Coe@PdoConfig* are set to TRUE.

7.2 Example for bus coupler

Reference File with example: ETG2001_ModulesSlots_BusCoupler.xml

Reference File with example for Modules: ETG2001_ModulesSlots_BusCoupler_Modules.xml

This example describes a bus coupler with the following features:

- Connects proprietary backplane bus terminals to the EtherCAT network
- Bus terminals are of different classes, analog in, analog out, digital in, digital out.

- Up to 254 terminals can be connected to the coupler
- Independent from their physical order analog terminals are mapped in the process data image in front of digital terminals
- The coupler itself has fixed RxPDO and TxPDO

7.2.1 Used ESI elements

The used ESI elements and attributes for this example are listed by Table 7.

Table 7: Used Element and Attributes

Element Name
Module:Type
Module:Type@ModuleIdent
Module:Type@ModuleClass
Module:Name
Module:RxPdo:Index@DependOnSlot
Module:RxPdo:Entry:Index@DependOnSlot
Module:TxPdo:Index@DependOnSlot
Module:TxPdo:Entry:Index@DependOnSlot
Slots:Slot@MinInstances
Slots:Slot@MaxInstances
Slots:Slot:Name
Slots:Slot:ModuleClass:Class
Slots:Slot:ModuleClass:Name
Slots:ModulePdoGroup

7.2.2 Coupler Process Data

The bus coupler itself has its own status (0xF100) and control (0xF200) word which are always available. The PDO indexes are set to the end of the index range for RxPDOs and TxPDOs.

This leaves a maximum of 254 (e.g. 0x1600 to 0x16FE) PDO indexes for possible connectable modules.

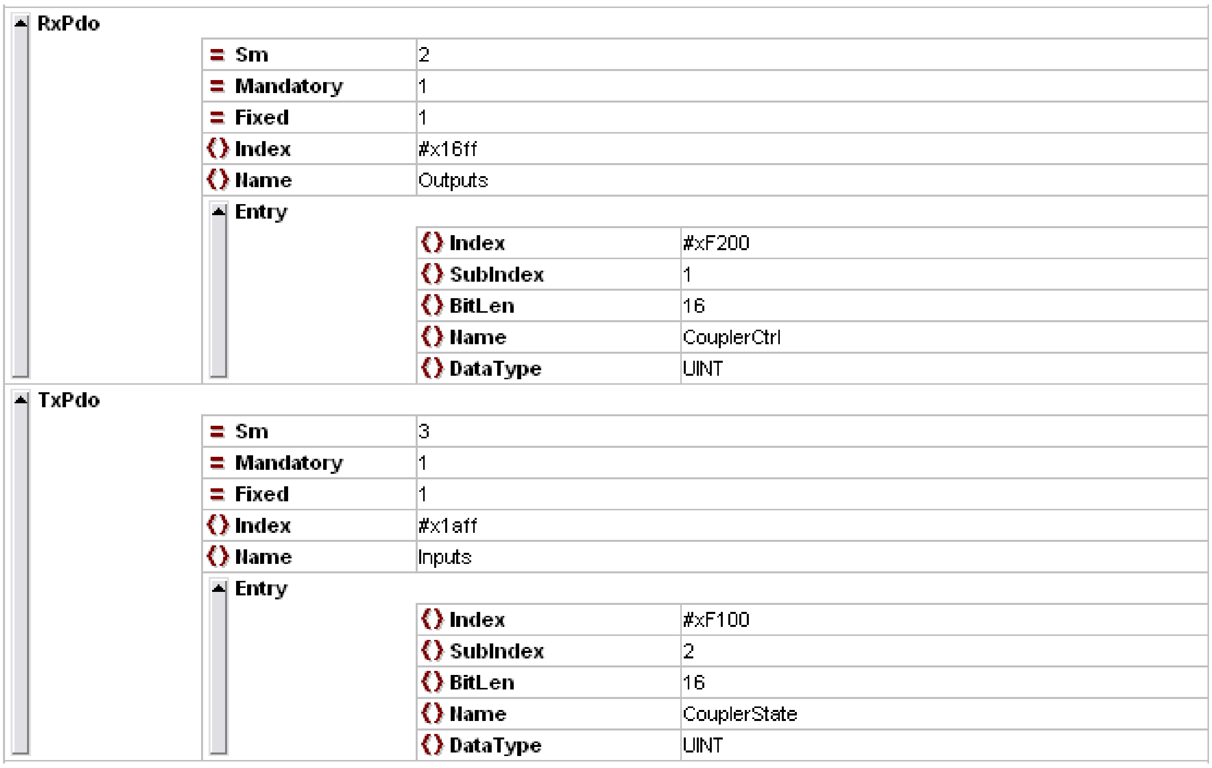


Figure 11: Description of Coupler Process data

7.2.3 Description of Modules

Each module, resp. each terminal has its own *ModuleIdent* which is used by the configuration tool to determine the module identity when reading the list of connected terminals from the bus coupler, i.e. reading back the “Detected Module Ident List” 0xF050 (see 0x9nn0:0A) in this example. It is not used by the configuration tool for the PDO mapping or PDO assign.

From the many possible bus terminals which can be connected to the coupler one of each ModuleClass is shown by Figure 12. The Modules are assigned to a *ModuleClass* which is described in the element *Slots:Slot:ModuleClass*.

For a compact process image terminals with byte-bordering process data are sorted to the beginning of the image, while devices having bit-granular process data (like digital terminals) are sorted to the end of the process data image.
To achieve this modules are assigned to different *ModulePdoGroups*: Digital devices are of *ModulePdoGroup* 2 and analog devices of *ModulePdoGroup* 1. The PDOs of the bus coupler itself are assigned to *ModulePdoGroup* 0 by the attribute *Device:Type@ModulePdoGroup*, i.e. they are the first variables in the process image.

The *Text* in element Type holds the name of the module, whereas the element *Name* a more detailed description of the module provides.
The Name can be provided in different languages. In this example the only language supported is English (language code ID = 1033).

Module (4)

	Type	Name	RxPdo	TxPdo
1	Type ModuleIdent #x00008201 ModuleClass kl_dig_in ModulePdoGroup 1 Text KL 1002	Name LcId 1033 Text KL 1002, 2 Ch. Input (24V, 3.0ms)		TxPdo Fixed 1 Sm 3 Index DependOnSlot=1 Name LcId=1033 Entry (2)
2	Type ModuleIdent #x00008202 ModuleClass kl_dig_out ModulePdoGroup 1 Text KL2012	Name LcId 1033 Text KL 2012, 2 Ch. Output (24V, 0.5 A)	RxPdo Fixed 1 Sm 2 Index DependOnSlot=1 Name LcId=1033 Entry (2)	
3	Type ModuleIdent #x00000bb9 ModuleClass kl_ana_in_x ModulePdoGroup 0 Text KL 3001	Name LcId 1033 Text KL 3001, 1 Ch. ana. Input (-10V/0...10V)	RxPdo Fixed 1 Sm 2 Index DependOnSlot=1 Name LcId=1033 Entry (3)	TxPdo Fixed 1 Sm 3 Index DependOnSlot=1 Name LcId=1033 Entry (3)
4	Type ModuleIdent #x00000fa1 ModuleClass kl_ana_out_x ModulePdoGroup 0 Text KL 4001	Name LcId 1033 Text KL 4001, 1 Ch. ana. Output (0...10V)	RxPdo Fixed 1 Sm 2 Index DependOnSlot=1 Name LcId=1033 Entry (3)	TxPdo Fixed 1 Sm 3 Index DependOnSlot=1 Name LcId=1033 Entry (3)

Figure 12: Modules description for bus terminals

The indexes of the PDOs are specified as “*DependOnSlot*” which means that the PDO index is adapted depending on the *Slot* number to which it is assigned to. The same applies for the PDO entries.

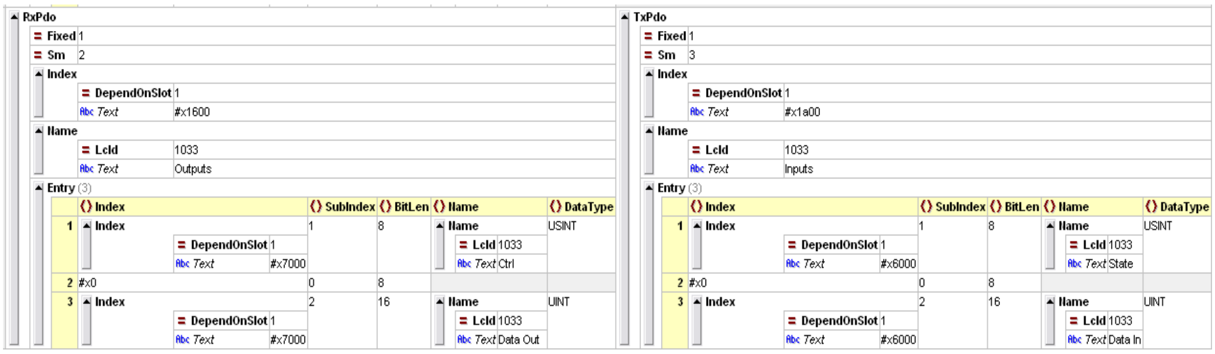


Figure 13: Modules PDO description for one bus terminal

7.2.4 Description of Slots

The coupler's *Slots* description is identical for all 254 slots. The *Slots* are named "Terminals" in the configuration tool according to the element *Slot:Name*. So the prototype of a slot is specified and can be instantiated between 0 and 254 times (*MinInstances*, *MaxInstances*).

In the configuration tool all 254 slots are represented by the *Slot:Name* element "Terminals".

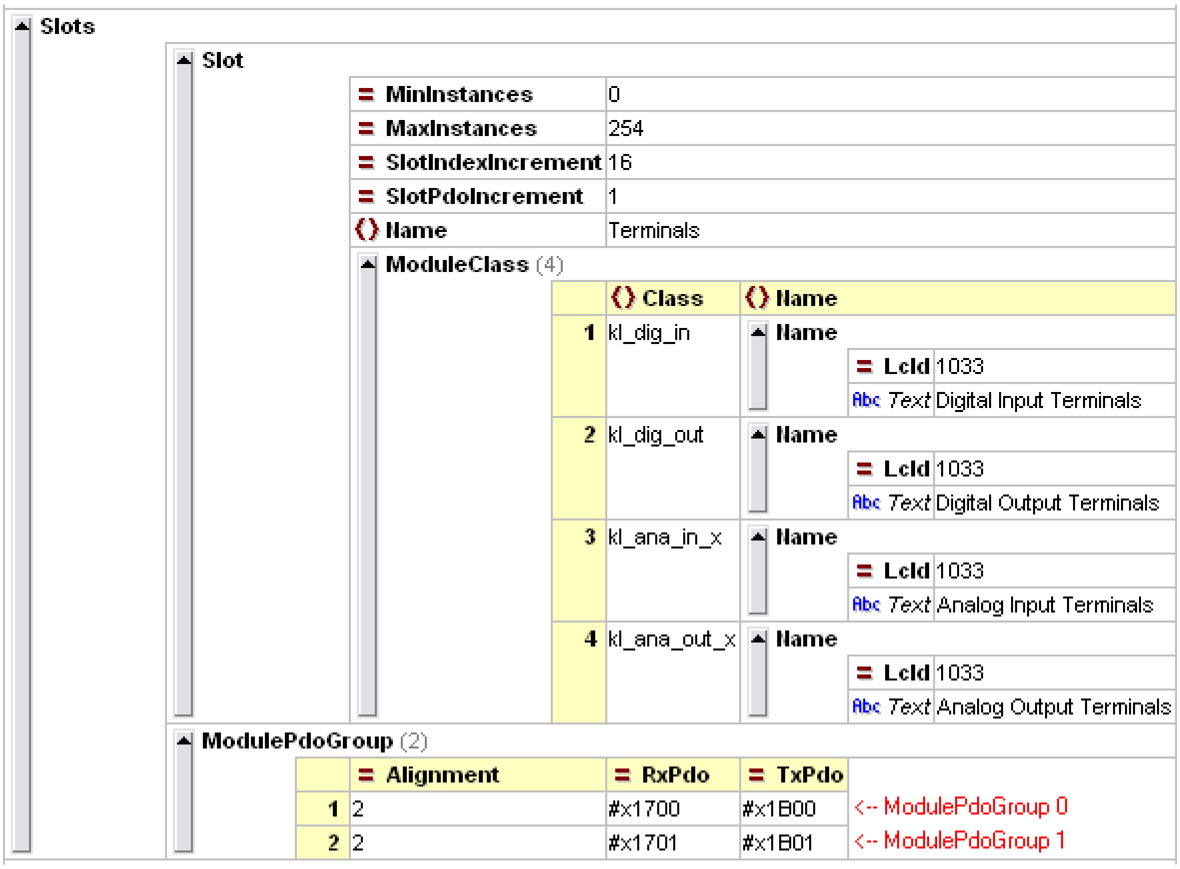


Figure 14: Slots Description

The *ModuleClasses* are used to sort the modules in the configuration tool. In this example there are four classes. Each class has one class member, i.e. one representative terminal. There could be many more for each Module class, i.e. many different digital input terminals.

Which modules (terminals) can be assigned to a *Slot* is described the list of ModuleClasses: all modules belonging to one of the listed classes can be assigned.

The order of the process data within the process image is already defined by the attribute *Modules:Module:Type@ModulePdoGroup*. The alignment between the *ModulePdoGroups* is yet to be done: The *Alignment* attribute defines the alignment to 2 bytes (word alignment). The PDOs used for the alignment of ModulePdoGroup 0 (the coupler itself) are 0x1702 and 0x1B02,

For the alignment of ModulePdoGroup 0 (analog devices) the objects 0x1700 and 0x1B00, and for ModulePdoGroup 1 (digital devices) the PDOs 0x1701 and 0x1B01 are used.
The alignment of the coupler's PDOs which are describe in the elements *RxPdo* and *TxPdo* the alignment would have to be done directly by inserting padding bits (element *Index* = 0, *SubIndex* = 0, *BitLen* = number of padding bits).

An example for a possible mapping and the corresponding PDO indexes and PDO entry indexes is shown by Table 8. The physical order of the terminals is reflected by the PDO and PDO entry numbers.

Table 8: PDO Index and PDO Entry Index Increment

Physical Order of Terminals	RxPDO /TxPDO		RxPDO / TXPDO Entries	
1 (KL 3001)	0x16**00**	0x1A**00**	0x70**00**	0x60**00**
	↓	↓	↓	↓
	SlotPdoIncrement = 1		SlotIndexIncrement = 16 (0x10)	
	↓	↓	↓	↓
2 (KL1002)	--	0x1A**01**	--	0x6**010**
	↓	↓	↓	↓
	SlotPdoIncrement = 1		SlotIndexIncrement = 16 (0x10)	
	↓	↓	↓	↓
3 (KL4001)	0x16**02**	0x1A**02**	0x7**020**	0x6**020**

The PDO configuration and PDO assignment is automatically generated by the configuration tool depending on the users modules selection. This configuration has to be downloaded to the slave which is done when the attributes *Mailbox:Coe@Assign* and *Mailbox:Coe@PdoConfig* are set to TRUE.

8 Attribute Virtual

8.1 Virtual flag for element PdoType only

The used ESI elements and attributes for this example are listed by Table 9.

Table 9: Used Element and Attributes

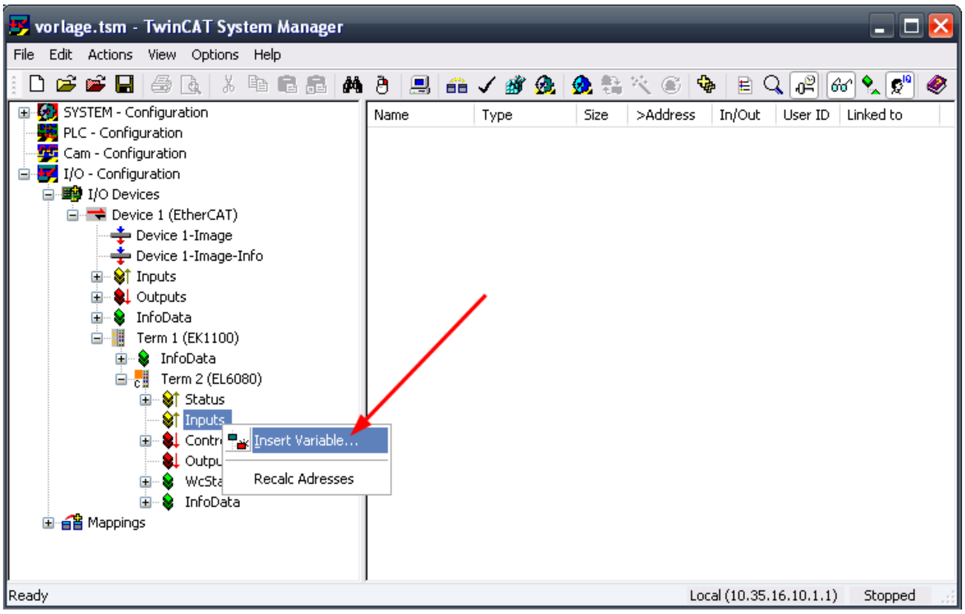
Element Name
PdoType@Virtual

In some cases it might be useful to allow a configuration tool to generate the Object Dictionary and the PDO entries of a slave. So the end-user has not to generate an OD entry and, as a result, receives a new variable but the other way around.

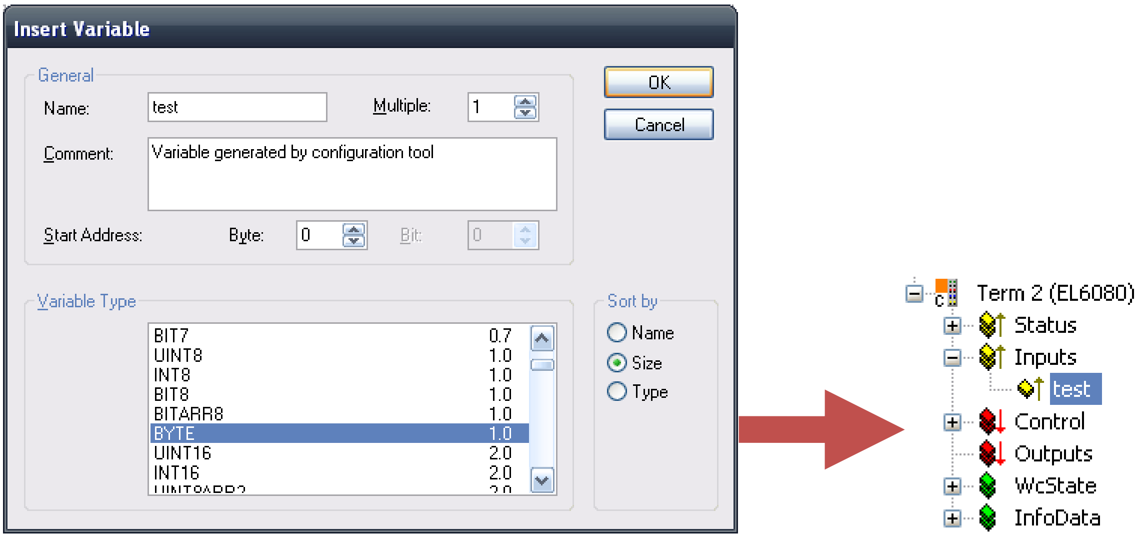
The attribute “virtual” for the elements Rx/TxPDO allows this. It is a placeholder for any data structure the Configuration Tool generates.

Of course the Configuration Tool also has to support this feature. The corresponding PDO entry AND the OD entry holding the data then can be downloaded to the slave (using PDO assign). The object than can be generated using any data type.

Example (Beckhoff System Manager):



The picture above shows a configuration tool which enables a user to add any variable to “Inputs”. The used decides which format and size:



8.2 Virtual flag for elements *PdoType* and *Sm* (Sync manager)

The used ESI elements and attributes for this example are listed b yTable 10.

Table 10: Used Element and Attributes

Element Name
Device:Sm@Virtual
PdoType@Virtual

The virtual flag of the element *Sm* is used if the functionality of a Sm is needed for configuration but no hardware entity has to be used.

This may be used id ESC register values are mapped into process data (TxPDO, RxPDO) of a slave. As register access always is consistent the Sm hardware is not needed to handle the memory access. Although a Sync manager it is needed for configuration within the ESI file to be able to assign the PDO entries to a Sm element.

The PDOs using this Sm shall be marked as virtual, too.

Example:

The next “Sync1 Start Time” shall be mapped into the process data. Register 0x998 holds this value and so this register has to be mapped into the TxPDO.

Sm

Sm (3)

	StartAddress	Control Byte	Enable	Virtual	Text
1	#x1000	#x44	1		Outputs
2	#x1100	#x20	0		Inputs
3	#x0998			true	Inputs

TxPdo

TxPdo

Fixed	1
Sm	2
Su	1
Virtual	true
Index	#x1a81
Name	NextSync1Time
Entry	

Sm 0 and 1 (entry 1 and 2) are used for normal process data. Sm 2 is used for mapping the register value.

So the start address is set to 0x998. Now a TxPdo entry can be created using this Sm.

There also can be an Fmmu assigned to this Sm. The register value than can be accessed by logical addressing. This can be done by defining a corresponding Fmmu entry in the ESI file.

Fmmu

Fmmu (2)

	Sm	Su	Text
1	0	0	Outputs
2	2	1	Inputs

9 Object Dictionary (element *Profile*)

To describe the CoE or SoE dictionary of a slave offline the element Profile is used. It is sperarated in 3 parts: the profile type, the data type definition area, and the object description itself.

The offline dictionary is used for elements holding useful data for a configuration tool for offline configuration (if the network is not connected and the configuration tool cannot access the online Dictionary).

All data types, simple (i.e. base data types) and complex (i.e. record, array) types which are used by any object are defined in the *DataType* element.

The objects are described in the *Objects* element. If the *Objects* element is supported, the mandatory objects (communication specific object in the area 0x1000 – 0x1FFF) have to be described. Device specific objects may be described.

9.1 Data Type description

Base data types (INT, UDINT, STRING(n), ..) are defined in ETG.1020 chapter “Base Data Types).

9.1.1 Object of simple data type (only SI0)

Object 0x1000 is a variable of data type UDINT and therefore only has one entry (subindex 0).

Also base data types have to be defined in the *Profile*:*DataTypes* element. The object 0x1000 itself is described in the *Object* element, the data type refers to the element in *DataTypes* with *Name* = “UDINT”.

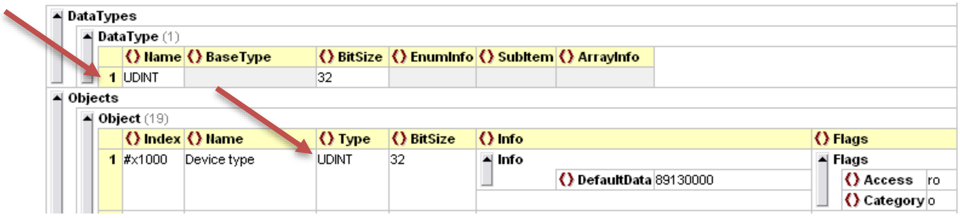


Figure 15: Object of variable data type

9.1.2 Objects of complex type (i.e. SI0, SI1-n)

Objects which have several sub-indexes have the structure as describe in Table 11.

Table 11: Structure of object with several SIs

General description		Example
Index	**Object index**	**Index 1018**
SubIndex 0	number of highes SubIndex	SI0 = 0x04
Subindex 1-n	entries of this object	SI1 = [Vendor ID]
		SI2 = [Product Code]
		SI3 = [Revision No]
		SI4 = [Serial No]

When subindexes 1-n are of different data types this object is of data type RECORD.

When subindexes 1-n are of the same data type of this object is of data type ARRAY.

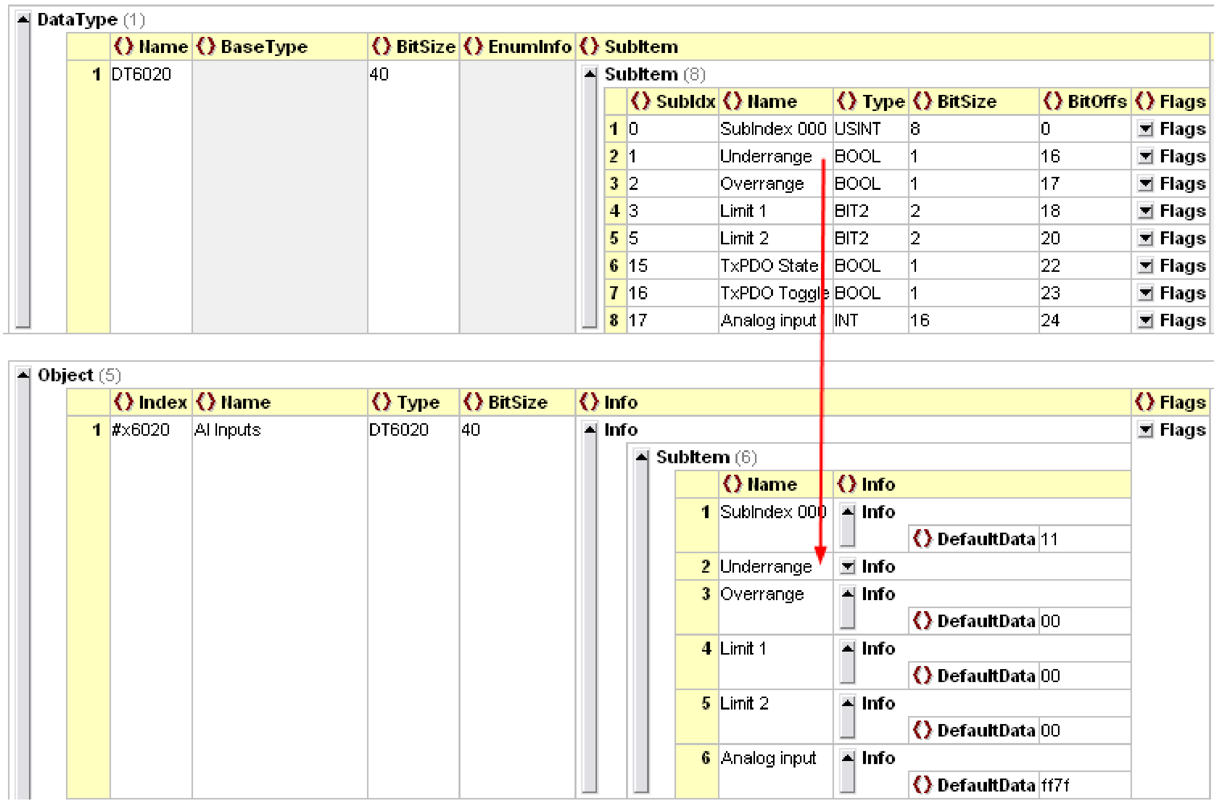
The data types of the object in this case always relate to the objects without considering SI0 (always data type USINT).

9.1.2.1 Si1-n have different data types

Figure 16 shows how the object 0x6020 is described. Since the object entries are of different data types the object is of type RECORD.

The data type of 0x6020 is named DT6020. To describe the object entries (SI0-SI17) the element *DataType:SubItem* is used.

Since the element *DataType:SubItem:Name* identifies the element *Objects:Info:SubItem:Name* they have to be identical (case sensivite).



DataType (1)

	Name	BaseType	BitSize	EnumInfo	SubItem
1	DT6020		40		SubItem (8)

	SubIdx	Name	Type	BitSize	BitOffs	Flags
1	0	SubIndex 000	USINT	8	0	Flags
2	1	Underrange	BOOL	1	16	Flags
3	2	Overrange	BOOL	1	17	Flags
4	3	Limit 1	BIT2	2	18	Flags
5	5	Limit 2	BIT2	2	20	Flags
6	15	TxPDO State	BOOL	1	22	Flags
7	16	TxPDO Toggle	BOOL	1	23	Flags
8	17	Analog input	INT	16	24	Flags

Object (5)

	Index	Name	Type	BitSize	Info	Flags
1	#x6020	AI Inputs	DT6020	40	Info / SubItem (6)	Flags

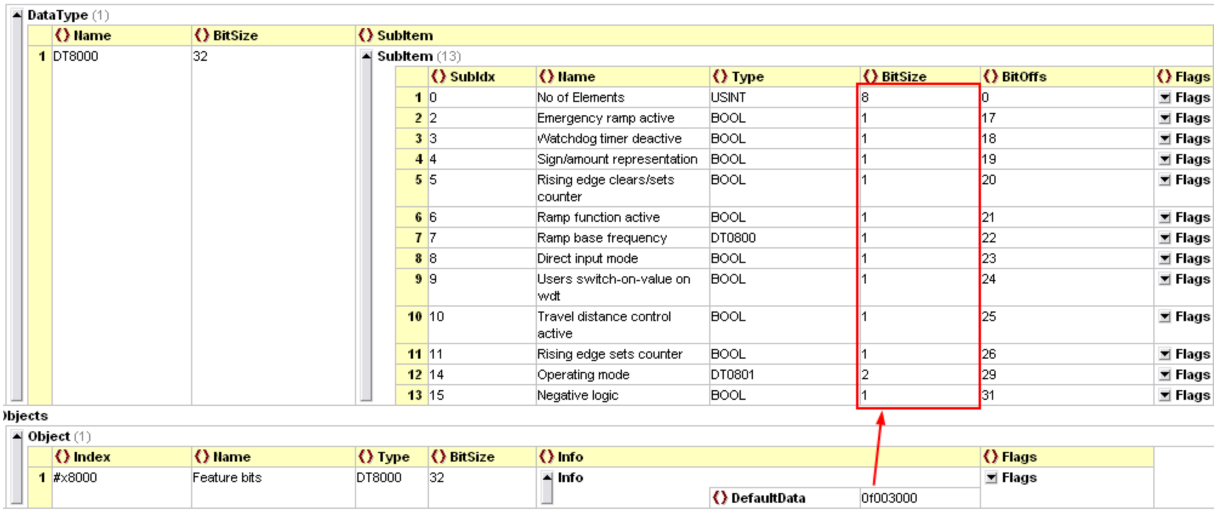
	Name	Info
1	SubIndex 000	Info: DefaultData 11
2	Underrange	Info
3	Overrange	Info: DefaultData 00
4	Limit 1	Info: DefaultData 00
5	Limit 2	Info: DefaultData 00
6	Analog input	Info: DefaultData ff7f

Figure 16: Names are references for subindexes for data type and object; single Default values

The default values of each *SubItem* can be described in the element *Object:SubItem:Info:DefaultData* as shown in Figure 16.

A more compact way to describe the default value of one complete object is shown by Figure 17. The Object does not describe each entry again rather than defiing the default value including SI0 as “byte sausage”.

In this case additional SubItem:Info values cannot be added any more.



DataType (1)

	Name	BitSize	SubItem
1	DT8000	32	SubItem (13)

	SubIdx	Name	Type	BitSize	BitOffs	Flags
1	0	No of Elements	USINT	8	0	Flags
2	2	Emergency ramp active	BOOL	1	17	Flags
3	3	Watchdog timer deactive	BOOL	1	18	Flags
4	4	Sign/amount representation	BOOL	1	19	Flags
5	5	Rising edge clears/sets counter	BOOL	1	20	Flags
6	6	Ramp function active	BOOL	1	21	Flags
7	7	Ramp base frequency	DT0800	1	22	Flags
8	8	Direct input mode	BOOL	1	23	Flags
9	9	Users switch-on-value on wdt	BOOL	1	24	Flags
10	10	Travel distance control active	BOOL	1	25	Flags
11	11	Rising edge sets counter	BOOL	1	26	Flags
12	14	Operating mode	DT0801	2	29	Flags
13	15	Negative logic	BOOL	1	31	Flags

Objects

Object (1)

	Index	Name	Type	BitSize	Info	Flags
1	#x8000	Feature bits	DT8000	32	Info: DefaultData 0f003000	Flags

Figure 17: Default values as byte sausage

9.1.2.2 SI1-n have same data type

The Sync manager object 0x1C12 is of data type ARRAY since SI1 to SIn are of data type UINT, i.e. identical data type. In thise case the data type (DT1C12) of the object is assembled as described in

Figure 18: Subidex 0 is described by *DataType:SubItem* element 1, the following entries are described by an ARRAY data type (DT1C12ARR). Each single array element describs another object entry of 0x1C12.

DataType:ArrayInfo:Elements is the amount of array elements The element *DataType:SubItem:SubIdx* remains empty.

When using ARRAY data type for an object no individual names for each object entry can be used. The object entry name has to be “SubIndex xxx”, while xxx is the decimal number of the subIndex and the blank inbetween has to be considered (see also *ObjectInfoType:SubIndex:Name*).

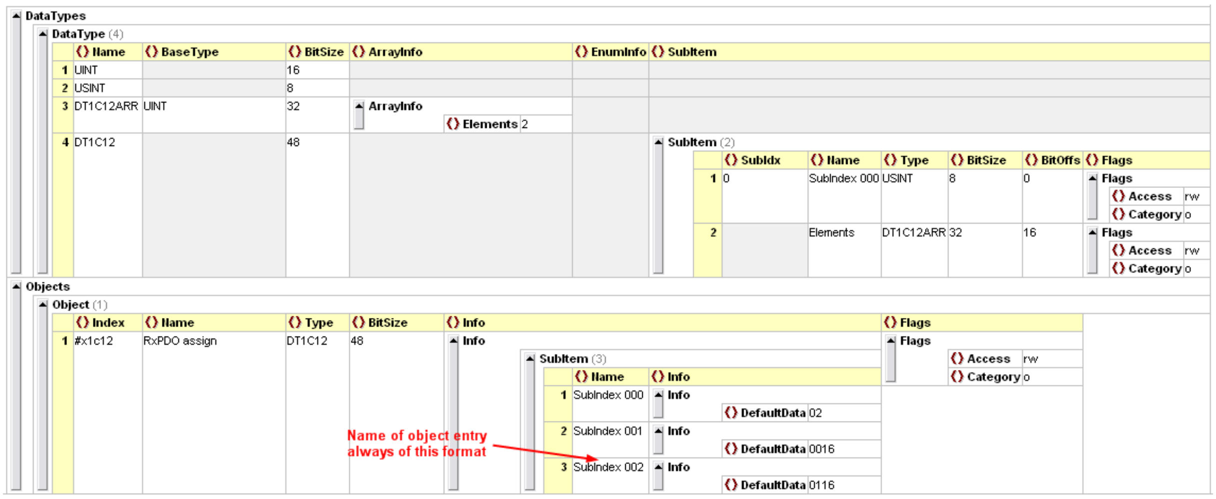


Figure 18: DataType and Object definition for complex object using ARRAY data type

9.1.3 Objects with ENUM data type

An ENUM allows users to select some options as easily readable text, and at the same time allows EtherCAT and the device to handle a machine-readable number. ENUMs are used as dropdown option in most cases.

In this example objects 0x2000 and 0x2001 are variables of ENUM data type. This data type is an enumeration of “constants” and each constant is represented by a “value”.

An ENUM is realized by a list (enumeration) of sequences of bytes. Each entry of this list, i.e. one sequence, is a combination of 4 bytes “value” (UDINT) and n following bytes “constant” (STRING(n)).

0x00 0x00 0xAF 0xFE	0x48 0x65 0x6c 0x6c 0x6f
Value (UDINT)	Constant (STRING(5))
45054	Hello

Figure 19: Masking ENUM entry data

The example in Figure 19 shows the sequence of one ENUM entry. The value “45054” represents the constant “Hello”.

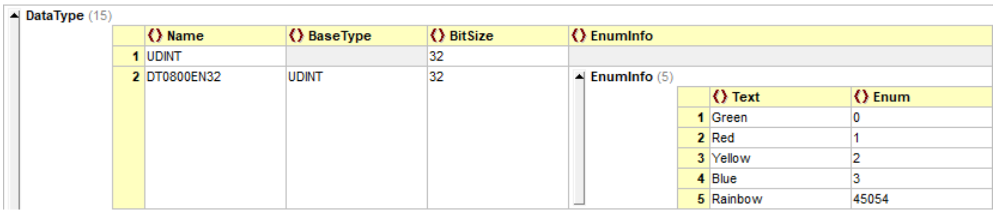
In the CoE Object Dictionary (OD) the ENUM area is from 0x800 to 0xFFF. Each index represents one ENUM, with the index number as CoE data type, and each subindex one entry (option) of the ENUM.

Note: as the value is interpreted as UDINT the byte ordering of that part is little endian during the transmission. If a master requests one entry (subindex) the slave would write 0xFE 0xAF 0x00 0x00

9.1.3.1 ENUM definition

As every data type used in the ESI Offline Dicitionary, ENUMs need to be defined in the element *Dictionary*:*DataTypes*.

Figure 20 shows an example of an ENUM definition with five entries:


DataType (15)

	Name	BaseType	BitSize	EnumInfo
1	UDINT		32	
2	DT0800EN32	UDINT	32	EnumInfo (5)

EnumInfo (5)

	Text	Enum
1	Green	0
2	Red	1
3	Yellow	2
4	Blue	3
5	Rainbow	45054

Figure 20: Definition of an ENUM (A)

BaseType defines how the value in child element *Enum* is coded.

BitSize defines that 32 bits of the ENUM value is used

Name “DT0800EN32” is composed of the following parts:

DT: “DataType”

0800: CoE data type 0x800, each enum needs a unique number from 0x800-0xFFF

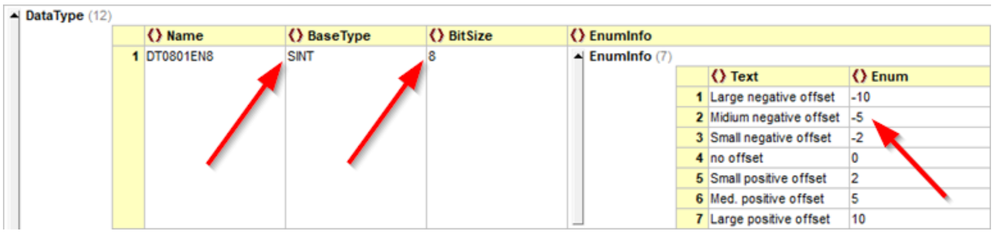
EN: “ENUM”

32: 32 bits of value used

Each *EnumInfo* entry represents one ENUM entry. Element *Text* defines the constant (text) and the element *Enum* defines the value.

Even if the value of the ENUM is specified as UDINT with 32 bits, the elements *BitSize* and *BaseType* may have different value.

In general, the value of an EMUN is not relevant for the user that selects the constant. The users select one textual option and then the device receives the corresponding value and evaluates it. In some cases, it might be useful that the value also is shown to the user. Therefor, each base data type with a length equal or less than 32 bit can be used to show the ENUM value. The element *BaseType* allows the value SINT for example. *BitSize* accordingly needs to be “8”. This allows to have values like “-5” in the element *EnumInfo:Enum*.


DataType (12)

	Name	BaseType	BitSize	EnumInfo
1	DT0801EN8	SINT	8	EnumInfo (7)

EnumInfo (7)

	Text	Enum
1	Large negative offset	-10
2	Midium negative offset	-5
3	Small negative offset	-2
4	no offset	0
5	Small positive offset	2
6	Med. positive offset	5
7	Large positive offset	10

Figure 21: Definition of an ENUM (B)

All values in element *Enum,* not based on UDINT (like “-5”), can be shown by a configuration tool and/or master that has access to the ESI file. The sequence of byte that is stored in the CoE object dictionary allways holds the casted UDINT value (0x000000FB instead of “-5”), which a master without ESI access would then display as 251.

9.1.3.2 ENUM useage as object data type

The ENUM data type then can be used in the object dictionary in the element *Dictionary:Objects*. It can be used as object and/or object entry data type. The object entry behaves like the base data type of the ENUM itself, i.e. the bit size of an object (entry) is the same as the bit size of the ENUM (element *BitSize*) namend in the element *Type*.

An ENUM also can be used instead of a specified data type as long as the bit size fits. For example, the object 0x6060 “modes of operation” is specified as SINT by the CiA. The ESI data type can be an ENUM with *BitSize* = “8” and *BaseType* = “SINT” instead (as in Figure 21). The user then can select a readable term instead of entering a number manually.

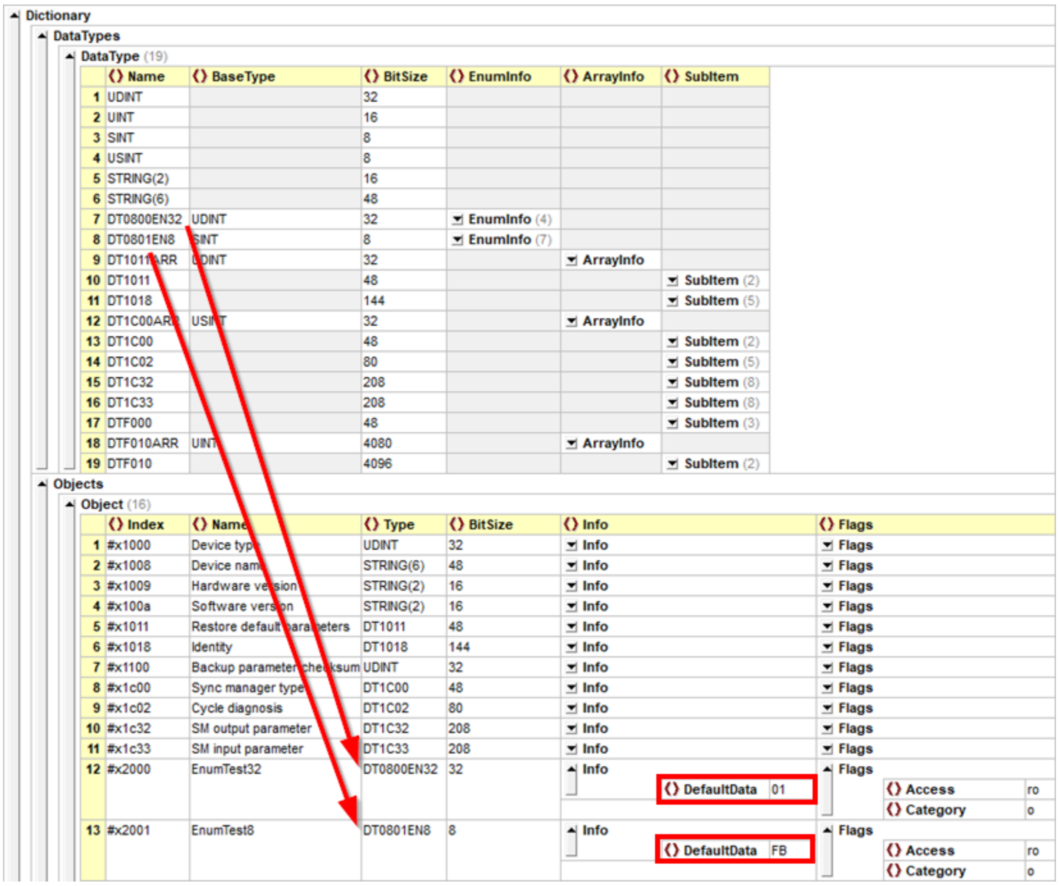


Figure 22: ENUM data type

Figure 22 shows the use of the two defined ENUMs (A and B). To preselect one ENUM option in the offline OD, the element *Info:DefaultData* can be used. To select “-5” in object 0x2001 by default, *DefaultData* needs to be “FB”.

Note: DefaultData describes the vaule if the EEPROM. So it is necessary to write the data in little endian. The default value of 45054 means default data = “FEAF0000”.

9.2 Flags

9.2.1 Object:Flags and DataType:SubItem:Flags

The used ESI elements and attributes for this example are listed by Table 12.

Table 12: Used Element and Attributes

Element Name
ObjectType:Flags:Access
ObjectType:Flags:Category
SubItemType:Flags:Access
SubItemType:Flags:Category

Example 1:

Object has simple data type, i.e. data type of object 0x1000 Device Type is UDINT.

For simple data types the access right, category, etc. for the object is described within the element *Object:Flags* as shown by Figure 23.

Dictionary

DataTypes

DataType (1)

	Name	BaseType	BitSize	EnumInfo	SubItem	ArrayInfo
1	UDINT		32			

Objects

Object (1)

	Index	Name	Type	BitSize	Info	Flags
1	#x1000	Device type	UDINT	32	Info: DefaultData 89130000	Flags: Access ro, Category o

Figure 23: Object with simple data type

Example 2:

Object has complex data type, i.e. object 0x1C32 SM Output Parameter has several sub-indices as shown in Figure 24 and Figure 25.

The access right and category described within element *Object:Flags* can only refer to the whole object. Since the object entries have different access rights (readonly or readwrite) they are described by the element *DataType:SubItem:Flags:Access*. In this case the access rights do not have to be described again in the element *Object:Flags* element.

Since default value of access is readonly anywas, the element *Flags:Access* would not be necessary to be described.

The category is also described within the element *DataType:SubItem:Flags*.

DataType (9)

	Name	BaseType	BitSize	EnumInfo	SubItem	ArrayInfo
1	DT1C32		176		SubItem (7)	

SubItem (7)

	SubIdx	Name	Type	BitSize	BitOffs	Flags: Access	Flags: Category
1	0	SubIndex 000	USINT	8	0	ro	o
2	1	Sync mode	UINT	16	16	rw	o
3	2	Cycle time	UDINT	32	32	rw	o
4	3	Shift time	UDINT	32	64	ro	o
5	4	Sync modes supported	UINT	16	96	ro	o
6	5	Minimum cycle time	UDINT	32	112	ro	o
7	6	Minimum shift time	UDINT	32	144	ro	o

Figure 24: Different access rights and category for object entries in element DataType

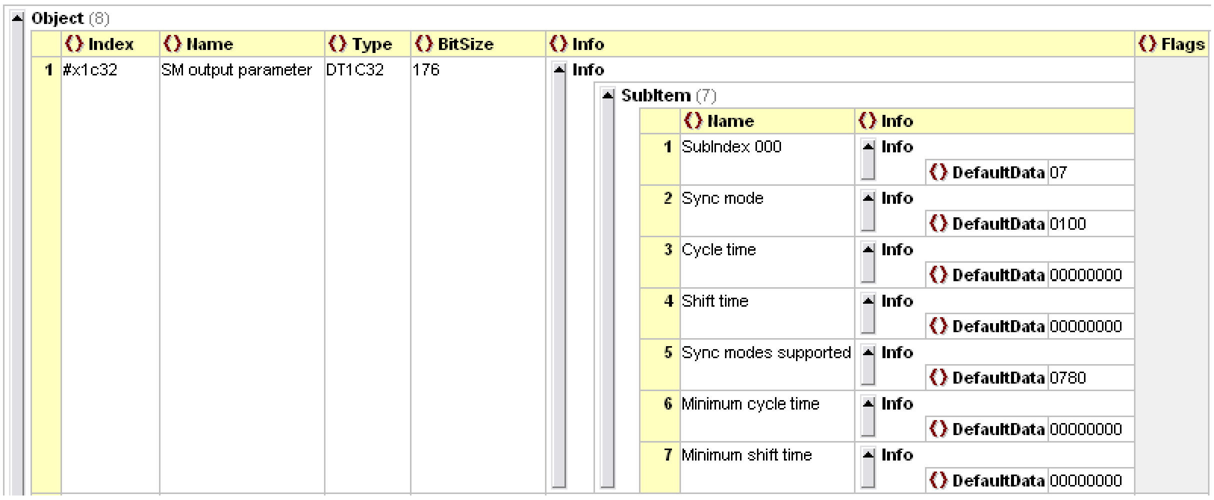


Figure 25: Different access rights and category for object entries in elemnt Object

9.3 Fixed, Mandatory, Exclude

The used ESI elements and attributes for this example are listed by Table 12.

Table 13: Used Elements and Attributes

Element Name
PdoType@Fixed
PdoType@Mandatory
PdoType:Exclude
PdoType:Entry@Fixed (EntryType@Fixed)

With MDP it is possible to configure the mapping objects and/or the content of the mapping objects in a flexible way.

The “fixed” and “mandatory” attributes are used to restrain the possibilities of editing PDO entries.

Example 1: device with two different PDOs

A device has the following process data:

```
0x7000:01 Input A
0x7000:02 Input B
```

```
0x7010:01 Input 1
0x7010:02 Input 2
0x7010:03 Input 3
```

The device shall always send Input value A and B and additionally the user shall choose if the combination of Input 1/3 or Input 2/3 shall be submitted within the process data.

This can be done by creating tree mapping PDOs.

```
PDO 0x1A00 containing 0x7000:01 and 0x7000:02 as PDO entries
PDO 0x1A01 containing 0x7010:01 and 0x7010:03 an PDO entries
PDO 0x1A01 containing 0x7010:02 and 0x7010:03 an PDO entries
```

The attribute “*mandatory*” of PDO 0x1a00 is set to true. So it is not possible to deselect this PDO (containing the two objects which always shall be submitted) in a configuration tool.

The *mandatory* flag of PDO 0x1a01 and 0x1a02 are set to false (or attribute is not available). So these two PDOs can be selected additionally in the configuration tool. To avoid that both, 0x1a01 and 0x1a02, PDOs are selected at the same time the element *Exclude* of 0x1a01 hold the value “0x1a02” and vice versa. If, for example, 0x1a01 is selected the configuration tool denies the possibility to select any PDO within the Exclude element of this entry (in this case 0x1a02).

The fixed element of all three PDOs is set to “true”. It is not possible to change the entry values in the configuration tool.

Example 2:

If the entries of one PDO itself should be editable using the configuration tool the fixed attribute of this PDO has to be set to false.

In this case the end user can add/delete/edit any entry of the PDO in the configuration tool.

For example the value 0x7030:02 can be added and 0x7010:03 can be deleted. If one single entry shall be fixed within the PDO the fixed attribute of this entry has to be true. This entry then is mandatory an can not be deleted.

9.4 PDO mapping data alignment

The used ESI elements and attributes for this example are listed by Table 12.

Table 14: Used Element and Attributes

Element Name
PdoType:Entry (EntryType)

The element of PdoType holds the “content” of an RxPDO or TxPDO element. Even the process data of a slave not using CoE is described by using indices and sub-indices.

How the process data configured in the PDO entries is submitted to the master is described within the following example:

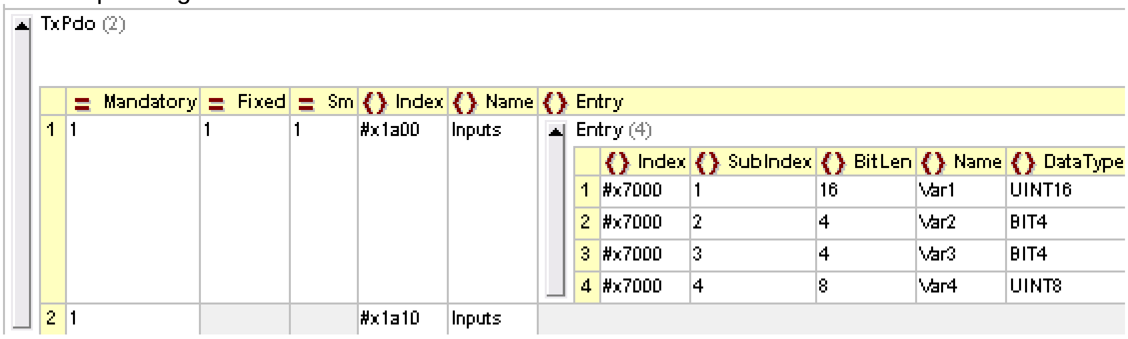
Example 1:

A Slave has the following process data:

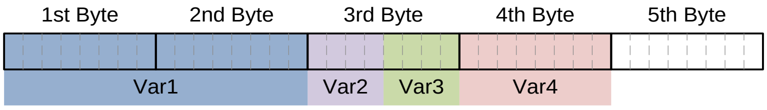
```
Var1: UINT16
Var2: BIT4
Var3: BIT4
Var4: UINT8
```

There are two possibilities to send the data to the master.

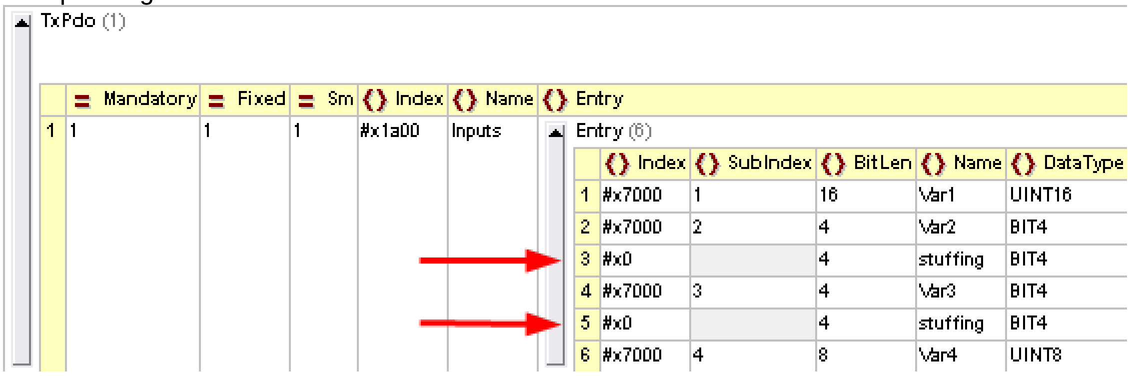
a) Without padding information:



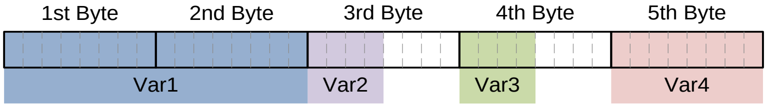
In this case the process data is send to the master in a consecutive way:



b) With padding Information:



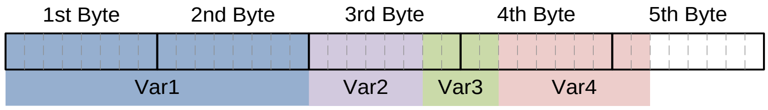
In this case the data is byte aligned:



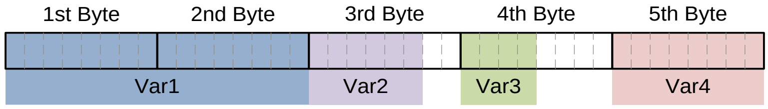
Example 2:

If Var2 has the data type BIT6 the process data also would be organized bit aligned.

Without padding information:



With padding information:



9.5 SubItem:DefaultData

9.5.1 Default Data for Complete Object

If the DefaultData (xs:HexBinary) are given for a complete object with several subindexes then they are represented in the same way as the SDO Complete Access Data as shown in Figure 24.

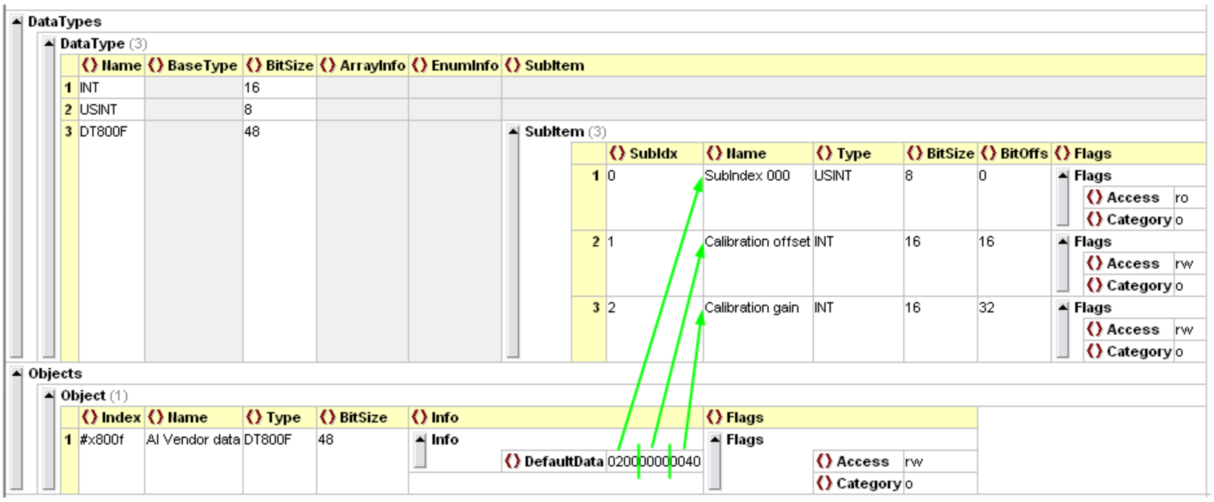


Figure 26: DefaultData for complete object

- Subindex0 shall be represented with 16 bit width
- representation is little endian

- the complete length of the default data has to match with the length of the object

The result looks as follows:

800F:0	AI Vendor data	RW	> 2 <
800F:01	Calibration offset	RW	0
800F:02	Calibration gain	RW	16384

9.5.2 Default Data for single Object Entries

The DefaultData can also be described for each individual object entry as shown in Figure 27.

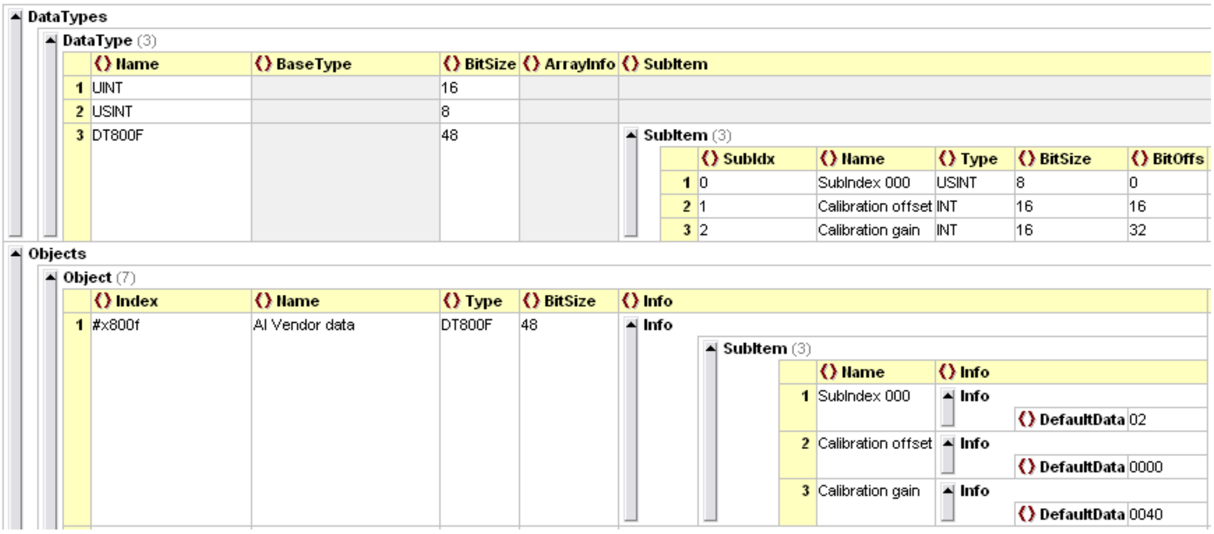


Figure 27: DefaultData for single entries

- SubIndex0 shall be described with 8 bit length
- Leading 0 (more significant bytes) may be omitted in the ESI file. They shall be added by the configtool

10 SubDevice

The element SubDevice enables one to group several ESC together. This might be useful if one device holds several ESCs. Normally each ESC would be shown as a separate device in a configuration tool.

With help of the SubDevice these devices can be grouped together in the ESI file. The configuration tool may then show one single device.

The first ESC (the ESC which is connected with port 0 to the IN port of the device) is called the “main device”. The ESCs following this first ESC are “subdevices”.

The element “SubDevice” of the main device holds a list of all other ESCs that belong to this device. The same element specifies to which port and ESC the actual Subdevice (ESC) is connected to.

Figure 28 shows the block diagram for an example:

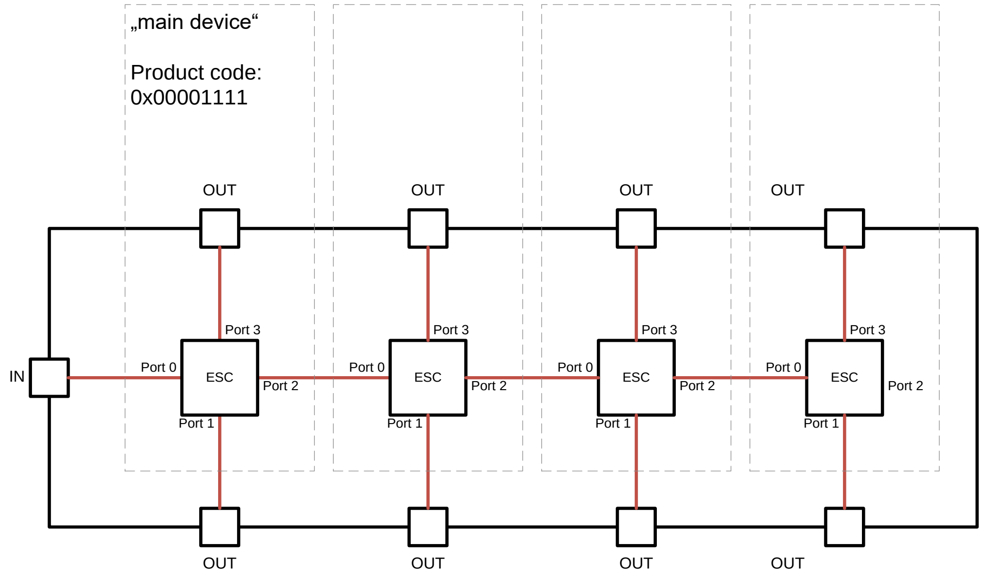


Figure 28: Block Diagram of Hardware with SubDevices

Here is an example to show how to uses SubDevice in ESI files.

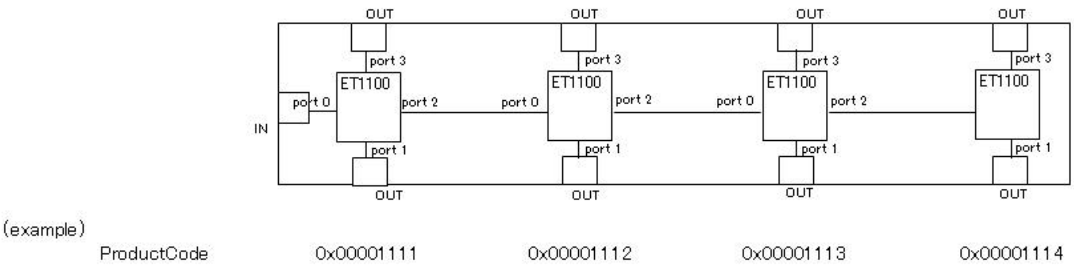


Figure 29: Block Diagram including Product Code Assignment

In the ESI the main device has a list with all SubDevices in the element SubDevice with their ProductCode and RevisionNo. The order top-down is also the physical order in which they are connected (Figure 30).

```
<Devices>
        <Device>
        <Type ProductCode="#x00001111"  RevisionNo="#x00000001"></Type>
        <SubDevice ProductCode="#x00001112"  RevisionNo="#x00000001" ></SubDevice>
        <SubDevice ProductCode="#x00001113"  RevisionNo="#x00000001" ></SubDevice>
```

<SubDevice ProductCode="#x00001114" RevisionNo="#x00000001" ></SubDevice>
</Device>

Figure 30: ESI of main device with reference to SubDevices

Figure 31 Shows the ESI of the SubDevices. The element PreviousDevice defines the outport to which the SubDevice is connected to. Again, the previous device is by definition the one which is listed in front of the current SubDevice.

<Device>
<Type ProductCode="#x00001112" RevisionNo="#x00000001"></Type>
<SubDevice PreviousDevice="0" PreviousPortNo="2"></SubDevice>
</Device>
<Device>
<Type ProductCode="#x00001113" RevisionNo="#x00000001"></Type>
<SubDevice PreviousDevice="1" PreviousPortNo="2"></SubDevice>
</Device>
<Device>
<Type ProductCode="#x00001114" RevisionNo="#x00000001"></Type>
<SubDevice PreviousDevice="2" PreviousPortNo="2"></SubDevice>
</Device>
</Devices>

Figure 31: ESI of SubDevices

Figure 32 shows another example of a main device with SubDevices.

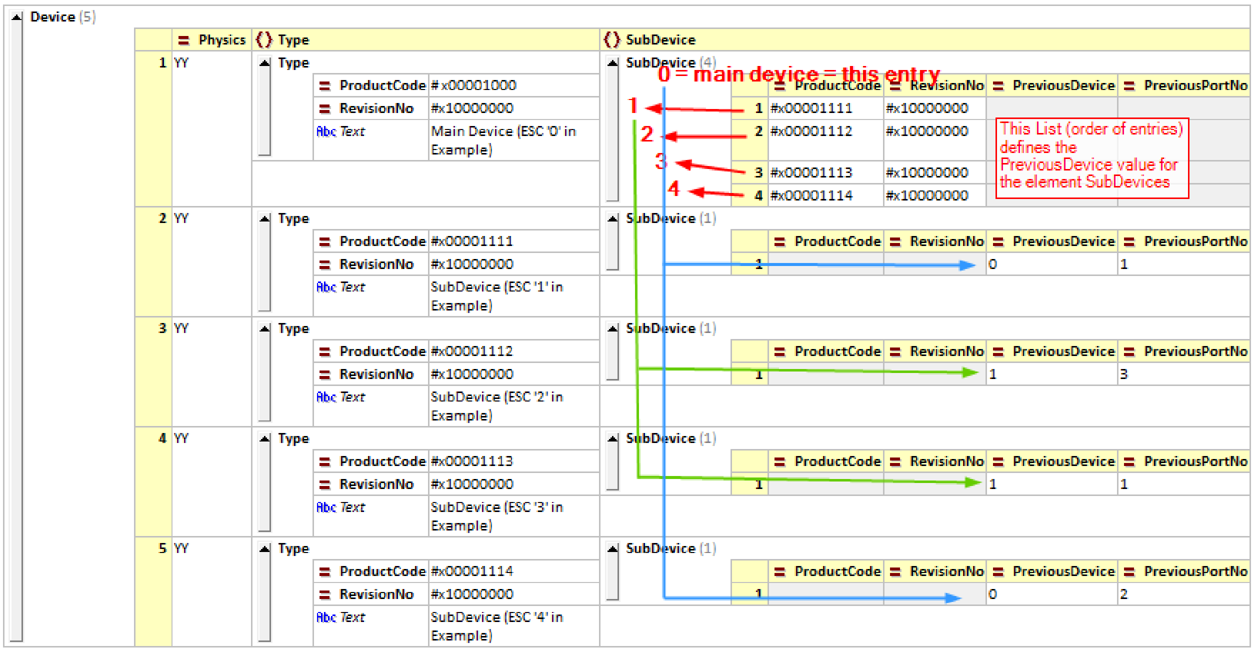


Figure 32: ESI of SubDevices