

Application Note ET9300 (EtherCAT Slave Stack Code)



Version 1.2
Date: 2011-11-16

LEGAL NOTICE

Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE® and XFC® are registered trademarks of and licensed by Beckhoff Automation GmbH. 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.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following German patent applications and patents: DE10304637, DE102004044764, DE102005009224, DE102007017835 with corresponding applications or registrations in various other countries.

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

© Beckhoff Automation GmbH 05/2009.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

DOCUMENT HISTORY

Version	Comment
1.0	Start document
1.1	Chapter 4 "Hardware access". Define descriptions; prototypes changed
1.2	Add chapter 5 Application Add chapter 9 Tool Add chapter 9 TestApplication Enhance Hardware access (chapter 4) update name references Enhance EoE chapter

NOTE: This document makes no claim to be complete regarding to the containing topics or the Slave Stack Code. For annotations or comments to this document please send an email to EthercatSSC@beckhoff.com.

CONTENTS

1	References	6
2	Terms, Definition, Abbreviation	7
	2.1 Abbreviation	7
3	Code Structure	8
4	Hardware Access	9
	4.1 Hardware Configuration	9
	4.2 Functions / Macros	9
	4.2.1 Generic	9
	4.2.2 Read Access	12
	4.2.3 Write Access	15
5	Application	18
	5.1 Calling Functions	18
	5.2 Functions	19
	5.2.1 Generic	19
	5.2.2 EtherCAT State Machine	19
	5.2.3 Process data handling	21
6	Objects	22
	6.1 Define local memory	22
	6.2 Object description	23
	6.3 Object name	24
	6.4 Object dictionary entry description	24
	6.5 Index Ranges	26
	6.6 Implementation examples	27
	6.6.1 Usage of Object Deftype ENUM	27
7	Mailbox	28
	7.1 FoE (File Transfer over EtherCAT)	28
	7.1.1 Testing FoE	28
	7.2 EoE (Ethernet over EtherCAT)	28
	7.2.1 Implementation	28
	7.2.2 EoE Examples	31
8	CiA402 drive profile	37
	8.1 Objects	37
	8.2 State machine	39
	8.3 Operation modes	40
	8.4 TwinCAT setup	41
	8.4.1 Automatic network setup	42
	8.4.2 Manual network setup	42
	8.4.3 NC parameter setup	44
9	TestApplication	45
	9.1 ESM Tests (0x2000 – 0x200F)	46

9.2	CoE Tests (0x2020 – 0x202F)	46
9.3	Generic Objects	46
10	Tool	48
10.1	Default Startup Dialogs	48
10.2	Main User Interface Elements	49
10.2.1	Tool Bar	49
10.2.2	Windows	51
10.3	Project Wizard	53
10.4	Create Files	54
10.5	Local SSC Update	55
10.6	Project Update	56
Appendix		58
	Support and Service	58
	Beckhoff's branch offices and representatives	58
	Beckhoff Headquarters	58
	Beckhoff Support	58
	Beckhoff Service	58
	EtherCAT Technology Group (ETG) Headquarters	58

FIGURES

Figure 1: EtherCAT slave stack.....	8
Figure 2: File-Stack Association.....	8
Figure 3: EL9800 Application object ranges.....	26
Figure 4: Send EoE datagram.....	30
Figure 5: Receive EoE datagram.....	31
Figure 6: EoE Example 1 (Schema).....	32
Figure 7: Network card settings.....	33
Figure 8: Access EtherCAT Slave Settings.....	33
Figure 9: EoE EtherCAT Slave Settings.....	34
Figure 10: Ping Command Window.....	34
Figure 11: EoE Example 2 (Schema).....	35
Figure 12: Enable IP Routing WinXP.....	35
Figure 13: Enable IP Routing WinCE.....	36
Figure 14: CiA402 state transitions and option codes.....	39
Figure 15: Axis configuration.....	41
Figure 16: Set device variable without PLC link.....	41
Figure 17: Scan for new EtherCAT devices.....	42
Figure 18: CiA402 axis setup.....	42
Figure 19: Add CiA402 device.....	43
Figure 20: Link multiple variables.....	43
Figure 21: Encoder scaling.....	44
Figure 22: Velocity scaling.....	44
Figure 23: Configurator Main User Interface.....	48
Figure 24: Configurator File Menu.....	49
Figure 25: Configurator Source Menu.....	49
Figure 26: Configurator Tool Menu.....	49
Figure 27: Configurator Options Window.....	50
Figure 28: Configurator Locked Setting.....	51
Figure 29: Configurator Help Menu.....	51
Figure 30: Configurator Project Information.....	52
Figure 31: Configurator File Context Menu.....	52
Figure 32: Configurator Slave Settings.....	53
Figure 33: Configurator Wizard Predefined Configurations.....	54
Figure 34: Configurator Create Files.....	55
Figure 35: Configurator Slave Stack Code Update.....	56
Figure 36: Configurator Project Update Dialog.....	56

TABLES

Table 1: Hardware Configuration	9
Table 2: TSDOINFOENTRYDESC member variables	23
Table 3: "TOBJECT" member variables	25
Table 4: Basic object index ranges	26
Table 5: Object definitions in file <code>cia402appl.h</code>	38
Table 6: State machine.....	40
Table 7: Linking of device and NC variables	43
Table 8: Test Object	45
Table 9: Test Object Entries	45
Table 10: Test Control Object.....	45
Table 11: Test Control Object Entries	45
Table 12: Test Object 0x2000 (ESM Group 1)	46
Table 13: Test Object 0x2020 (CoE Group 1).....	46
Table 14: Generic Objects.....	46

1 References

- [1] ETG.6010 Implementation Guideline for the CiA402 Drive Profile (V. 0.2.0)
- [2] ETG.1000 part 6 Application Layer protocol specification (V. 1.0.2)
- [3] Application Note EL9800 (V1.1)
- [4] ETG.2000 Slave Information Specification

2 Terms, Definition, Abbreviation

Base Datatypes -- CoE Datatypes defined in ETG.1000.6

Entry – in conclusion with object single element,
in conclusion with object dictionary the objects

Subindex -- describes a single element (entry) of an object

Object dictionary – the object dictionary is a list of objects. Within this list each object is uniquely identified by an (object) index.

2.1 Abbreviation

Abbreviation	Description
AL	Application Layer
CoE	CANopen application profile over EtherCAT CANopen™ is a registered trademark of CAN in Automation e.V., Nuremberg, Germany
CiA402	CANopen™ Drive Profile specified in IEC 61800-7-201; CANopen™ and CiA™ are registered trademarks of CAN in Automation e.V., Nuremberg, Germany
csp	cycle synchronous position
csv	cycle synchronous velocity
DC	Distributed Clocks
EoE	Ethernet over EtherCAT
ESC	EtherCAT Slave Controller
FoE	File Transfer over EtherCAT
GPO	General Purpose Output
NC	Numeric Control
PDI	Process data interface
PDO	Process Data Object
PLC	Programmable Logic Controller
SI	SubIndex
SM	Sync Manager
SSC	Slave Stack Code

3 Code Structure

The EtherCAT slave stack as seen in Figure 1 consists of three parts:

- Hardware access (4Hardware Access)
- Generic EtherCAT stack
- User Application

The functions and macros which shall be provided by the hardware access layer are defined in 4 Hardware Access. The behavior of the generic EtherCAT stack is described in ETG.1000 Specification [2] . The user defined functions are located in the files named with xxxappl.*.

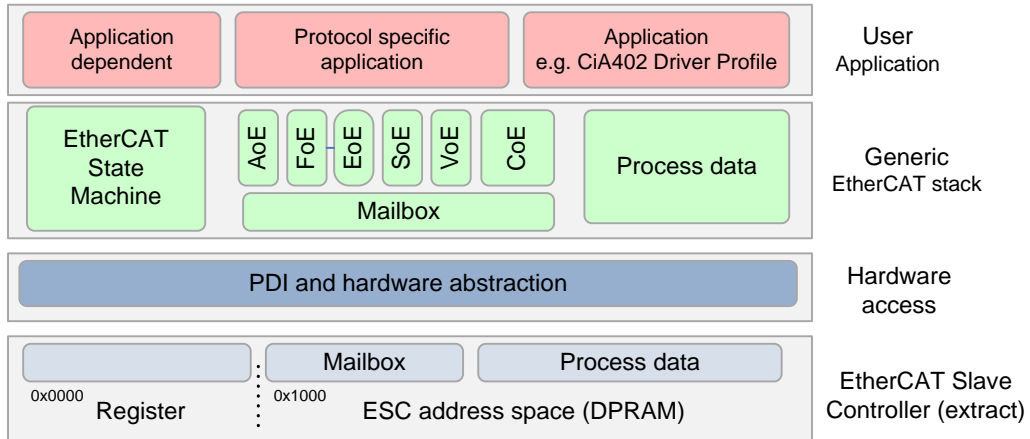


Figure 1: EtherCAT slave stack

Figure 2 shows the association between the Slave stack and the source files.

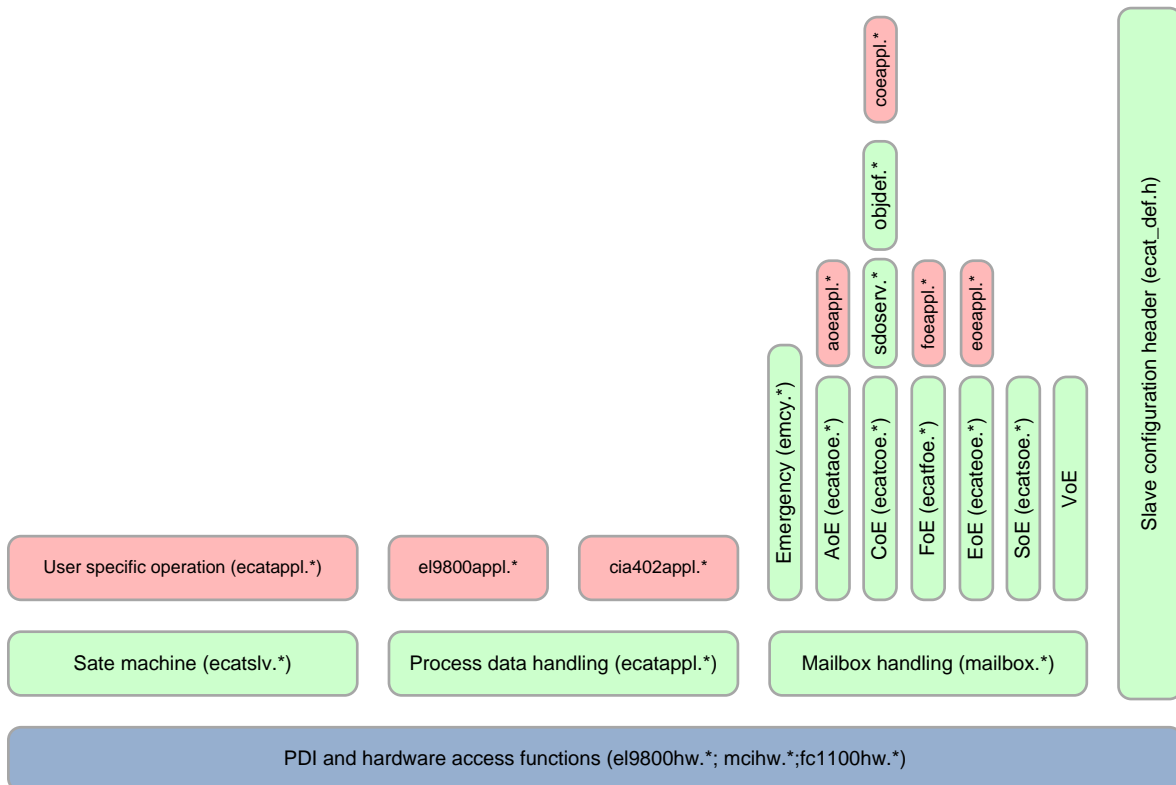


Figure 2: File-Stack Association

The structure of the code can be adapted to the application specific requirements by using the Slave Stack Code Tool (Error! Reference source not found. Error! Reference source not found.).

4 Hardware Access

The Slave Stack Code is executable on multiple platforms and controller architectures. Therefore the source code includes multiple defines to fulfill the specific hardware requirements. This chapter describes the hardware specific configurations and the hardware access functions which are used by the generic stack layer.

4.1 Hardware Configuration

The hardware related defines are described in Table 1.

Table 1: Hardware Configuration

Define	Description
BIG_ENDIAN_FORMAT	This define shall be set if the controller is working in Big Endian (Motorola) format. See also "BIG_ENDIAN_16BIT".
BIG_ENDIAN_16BIT	This define shall be set if the controller is working in Big Endian (Motorola) format and the switching of the high and low byte is done in hardware.
CONTROLLER_16BIT	Shall be set if the code is executed on a 16Bit architecture. If „BIG_ENDIAN_16BIT“ is set this define shall be reset.
CONTROLLER_32BIT	Shall be set if the code is executed on a 32Bit architecture. If „BIG_ENDIAN_16BIT“ is set this define shall be reset.
MEM_ADDR	This define is set to an 8/16 or 32 Bit variable depending on the selected architecture ("CONTROLLER_16BIT"; "CONTROLLER_32BIT").
ESC_16BIT_ACCESS	If this define is set the slave stack performs access to valid 16Bit ESC addresses only.
ESC_32BIT_ACCESS	If this define is set the slave stack performs access to valid 32Bit ESC addresses only.

4.2 Functions / Macros

4.2.1 Generic

Prototype: **UINT16 HW_Init(void)**

Parameter void

Return 0 if initialization was successful
 > 0 if error has occurred while initialization

Description Initializes the host controller, process data interface (PDI) and allocates resources which are required for hardware access.

Prototype: **void HW_Release(void)**

Parameter void

Return void

Description Release allocated resources.

Prototype: **UINT16 HW_GetALEventRegister(void)**

Parameter void

Return Content of register 0x220-0x221

Description Get the first two bytes of the AL Event register (0x220-0x221).

Prototype: **UINT16 HW_GetALEventRegister_Isr(void)**

Parameter void

Return Content of register 0x220-0x221

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as HW_GetALEventRegister.

Get the first two bytes of the AL Event register (0x220-0x221).

Prototype: **void HW_ResetALEventMask(UINT16 intMask)**

Parameter “intMask” Interrupt mask (disabled interrupt shall be zero)

Return void

Description Performs a logical AND with the AL Event Mask register (0x0204 : 0x0205). This function is only required if “AL_EVENT_ENABLED” is set.

Prototype: **void HW_SetALEventMask(UINT16 intMask)**

Parameter “intMask” Interrupt mask (enabled interrupt shall be one)

Return void

Description Performs a logical OR with the AL Event Mask register (0x0204 : 0x0205). This function is only required if “AL_EVENT_ENABLED” is set.

Prototype: **void HW_SetLed(UINT8 RunLed,UINT8 ErrLed)**

Parameter “RunLed” EtherCAT Run LED state

“ErrLed” EtherCAT Error LED state

Return void

Description Updates the EtherCAT Run and Error LEDs (or EtherCAT Status LED).

Prototype: **void HW_RestartTarget(void)**

Parameter void

Return void

Description Resets the hardware. This function is only required if “BOOTSTRAPMODE_SUPPORTED” is set.

Prototype: **void HW_DisableSyncManChannel(UINT8 channel)**

Parameter “channel” SyncManager channel

Return void

Description Disables selected SyncManager channel. Sets bit 0 of the corresponding 0x807 register.

Prototype: **void HW_EnableSyncManChannel (UINT8 channel)**

Parameter “channel” SyncManager channel

Return void

Description Enables selected SyncManager channel. Resets bit 0 of the corresponding 0x807 register.

Prototype: **TSYNCMAN * HW_GetSyncMan(UINT8 channel)**

Parameter “channel” SyncManager channel

Return Pointer to the SyncManager channel description. The SyncManager description structure size is always 8 Byte, the content of “TSYNCMAN” differs depending on the supported ESC access.

Description Gets the content of the SyncManager register from the stated channel. Reads 8 Bytes starting at 0x800 + 8*channel.

Prototype: **UINT16 HW_GetTimer(void)**

Parameter void

Return Current timer value

Description Reads the current register value of the hardware timer. If no hardware timer is available the function shall return the counter value of a multimedia timer. The counter increments per ms are defined with “ECAT_TIMER_INC_P_MS”. This function is only required if “ECAT_TIMER_INT” is not set.

Prototype: **void HW_ClearTimer(void)**

Parameter	void
Return	void
Description	Clears the hardware timer value. This function is only required if ECAT_TIMER_INT is not set.

Prototype: **UINT16 HW_EepromReload (void)**

Parameter	Void
Return	0 <> Error during EEPROM reload 0 = EEPROM load correct
Description	This function is called if a EEPROM reload request is triggered by the master. Only required if EEPROM Emulation is supported.

4.2.2 Read Access

Prototype: **void HW_EscRead(MEM_ADDR *pData, UINT16 Address, UINT16 Len)**

Parameter	"pData"	Pointer to local destination buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).
	"Len"	Access size in Bytes
Return	void	
Description	Reads from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.	

Prototype: **void HW_EscReadIsr(MEM_ADDR *pData, UINT16 Address, UINT16 Len)**

Parameter	"pData"	Pointer to local destination buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).
	"Len"	Access size in Bytes
Return	void	

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscRead".
Reads from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.

Prototype: **void HW_EscReadDWord(UINT32 DWordValue, UINT16 Address)**

Parameter "DWordValue" Local 32Bit variable where the register value shall be stored.
"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used..

Return void

Description Reads one word from the specified address of the EtherCAT Slave Controller.

Prototype: **void HW_EscReadDWordIsr(UINT16 DWordValue, UINT16 Address)**

Parameter "DWordValue" Local 32Bit variable where the register value shall be stored.
"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.

Return void

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadWord".
Reads one word from the specified address of the EtherCAT Slave Controller.

Prototype: **void HW_EscReadWord(UINT16 WordValue, UINT16 Address)**

Parameter "WordValue" Local 16Bit variable where the register value shall be stored.
"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.

Return void

Description Reads one word from the specified address of the EtherCAT Slave Controller. Only required if "ESC_32BIT_ACCESS" is not set.

Prototype: **void HW_EscReadWordIsr(UINT16 WordValue, UINT16 Address)**

Parameter "WordValue" Local 16Bit variable where the register value shall be stored.

	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.
Return	void	
Description	<p>This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadWord".</p> <p>Reads one word from the specified address of the EtherCAT Slave Controller. Only required if "ESC_32_BIT_ACCESS" is not set.</p>	
Prototype:	void HW_EscReadByte(UINT8 ByteValue, UINT16 Address)	
Parameter	"ByteValue"	Local 8Bit variable where the register value shall be stored.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes.
Return	void	
Description	<p>Reads one byte from the EtherCAT Slave Controller.</p> <p>Only required if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are not set.</p>	
Prototype:	void HW_EscReadByteIsr(UINT8 ByteValue, UINT16 Address)	
Parameter	"ByteValue"	Local 8Bit variable where the register value shall be stored.
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes.
Return	void	
Description	<p>This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscReadByte".</p> <p>Reads one byte from the EtherCAT Slave Controller.</p> <p>Only required if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are not set.</p>	
Prototype:	void HW_EscReadMbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)	
Parameter	"pData"	Pointer to local destination mailbox buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).
	"Address"	EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).
	"Len"	Access size in Bytes
Return	void	

Description Reads data from the ESC and copies to slave mailbox memory. If the local mailbox memory is also located in the application memory this function is equal to "HW_EscRead".

4.2.3 Write Access

Prototype: **void HW_EscWrite(MEM_ADDR *pData, UINT16 Address, UINT16 Len)**

Parameter "pData" Pointer to local source buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).

"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).

"Len" Access size in Bytes

Return void

Description Writes from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.

Prototype: **void HW_EscWriteIsr(MEM_ADDR *pData, UINT16 Address, UINT16 Len)**

Parameter "pData" Pointer to local source buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).

"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).

"Len" Access size in Bytes

Return void

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWrite".
Writes from the EtherCAT Slave Controller. This function is used to access ESC registers and the DPRAM area.

Prototype: **void HW_EscWriteDWord(UINT32 DWordValue, UINT16 Address)**

Parameter "DWordValue" Local 32Bit variable which contains the data to be written to the ESC memory area.

"Address" EtherCAT Slave Controller address. Specifies the offset within

the ESC memory area in Bytes. Only valid 32Bit addresses are used.

Return void

Description Writes one word to the EtherCAT Slave Controller.

Prototype: **void HW_EscWriteDWordIsr(UINT32 DWordValue, UINT16 Address)**

Parameter "DWordValue" Local 32Bit variable which contains the data to be written to the ESC memory area.

"Address" EtherCAT Slave Controller address . Specifies the offset within the ESC memory area in Bytes. Only valid 32Bit addresses are used.

Return void

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteWord".
Writes one word to the EtherCAT Slave Controller.

Prototype: **void HW_EscWriteWord(UINT16 WordValue, UINT16 Address)**

Parameter "WordValue" Local 16Bit variable which contains the data to be written to the ESC memory area.

"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.

Return void

Description Writes one word to the EtherCAT Slave Controller. Only required if "ESC_32BIT_ACCESS" is not set.

Prototype: **void HW_EscWriteWordIsr(UINT16 WordValue, UINT16 Address)**

Parameter "WordValue" Local 16Bit variable which contains the data to be written to the ESC memory area.

"Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid 16Bit addresses are used.

Return void

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteWord".
Writes one word to the EtherCAT Slave Controller. Only required if "ESC_32BIT_ACCESS" is not set.

Prototype: **void HW_EscWriteByte (UINT8 ByteValue, UINT16 Address)**

Parameter "ByteValue " Local 8Bit variable which contains the data to be written to the ESC memory area.

 "Address" EtherCAT Slave Controller address.Specifies the offset within the ESC memory area in Bytes.

Return void

Description Writes one byte to the EtherCAT Slave Controller.
Only defined if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are disabled.

Prototype: **void HW_EscWriteByteIsr(UINT8 ByteValue, UINT16 Address)**

Parameter "ByteValue" Local 8Bit variable which contains the data to be written to the ESC memory area

 "Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes.

Return void

Description This function should be implemented if a special function for ESC access from interrupt service routines is required; otherwise this function is defined as "HW_EscWriteByte".
Writes one byte to the EtherCAT Slave Controller.
Only defined if "ESC_16BIT_ACCESS" and "ESC_32BIT_ACCESS" are disabled.

Prototype: **void HW_Esc_Write_MbxMem(MEM_ADDR *pData, UINT16 Address, UINT16 Len)**

Parameter "pData" Pointer to local source mailbox buffer. Type of the pointer depends on the host controller architecture (specified in ecat_def.h or the SSC Tool).

 "Address" EtherCAT Slave Controller address. Specifies the offset within the ESC memory area in Bytes. Only valid addresses are used depending on 8Bit/16Bit or 32 Bit ESC access (specified in ecat_def.h or the SSC Tool).

 "Len" Access size in Bytes.

Return void

Description Writes data from the slave mailbox memory to ESC memory. If the local mailbox memory is also located in the application memory this function is equal to "HW_EscWrite".

5 Application

The Slave Stack Code contains a set of predefined applications including

- EL9800 EtherCAT Evaluation Kit application (8(4) Digital I/O, 16Bit Analog Input)
- CiA402 2Axis sample Implementation
- Test application
- Hardware independent sample application (fallback application if none of the previous is selected)

Each of these applications can be activated by setting the corresponding configuration defines in the file "ecat_def.h" or by selecting the configuration in the SSC Tool.

To create a new slave application, select either one of the predefined applications to adapt (e.g CiA 402 Drive) or disable all defined applications which enables a basic sample application which can be modified or enhanced.

If the application is created from the scratch a couple of functions need to be implemented and called. The prototypes of these functions are described in the following sub clauses.

5.1 Calling Functions

These functions are provided by the generic stack and shall be called from the application layer.

Prototype: **UINT16 MainInit(void)**

Parameter Void

Return 0 if initialization was successful
 > 0 if error has occurred while initialization

Description Initialize the generic slave stack.
 This function should be called after the platform including operating system and ESC is ready to use.

Prototype: **void MainLoop(void)**

Parameter Void

Return Void

Description This function handles the low priority function like EtherCAT state machine handling, mailbox protocols and if no synchronization is enabled also the application.
 This function shall be called cyclically from the application.

Prototype: **void ECAT_StateChange(UINT8 aIStatus, UINT16 aIStatusCode)**

Parameter aIStatus Requested AI Status
 aIStatusCode AL Status Code. (if != 0 the error flag indication will be set)

Return Void

Description This function shall be called by the application to trigger state transition in case of an error or to complete a pending transition.
 NOTE: state requests to a higher state than the current state are not allowed.

5.2 Functions

5.2.1 Generic

Prototype: **void APPL_Application(void)**

Parameter Void

Return void

Description This function is called by the synchronization ISR or from the mainloop if not synchronization is activated.

Prototype: **UINT16 APPL_GetDeviceID (void)**

Parameter Void

Return Explicit Device ID which is written to the AL Status Code register.

Description This function is called if the master requests the Explicit Device ID. Only required if the slave supports Explicit Device ID handling (EXPLICIT_DEVICE_ID).

5.2.2 EtherCAT State Machine

Each ESM function returns a 16Bit Value which reflects the result of the state transition.

Return value:

- 0 Indicates a successful transition.
- 0xFF Indicates a pending state transition (the application need to complete the transition by calling ECAT_StateChange).
- Other Indicates the reason for the failed transition. See [2] for a list of valid return codes.

Prototype: **UINT16 APPL_StartMailboxHandler(void)**

Parameter Void

Return See generic ESM return code description

Description This function is called during the state transition from INIT to PREOP or INIT to BOOT.

Prototype: **UINT16 APPL_StopMailboxHandler(void)**

Parameter Void

Return See generic ESM return code description

Description This function is called during the state transition from PREOP to INIT or BOOT to INIT.

Prototype: **UINT16 APPL_StartInputHandler (UINT16 *pIntMask)**

Parameter pIntMask Value for register 0x204 (AL Event Mask).

Return See generic ESM return code description

Description This function is called during the state transition from PREOP to SAFEOP (even if no input process data is available).

Prototype: **UINT16 APPL_StopInputHandler (void)**

Parameter Void

Return See generic ESM return code description

Description This function is called during the state transition from SAFEOP to PREOP(even if no input process data is available).

Prototype: **UINT16 APPL_StartOutputHandler (void)**

Parameter Void

Return See generic ESM return code description

Description This function is called during the state transition from SAFEOP to OP (even if no output process data is available).

Prototype: **UINT16 APPL_StopOutputHandler (void)**

Parameter Void

Return See generic ESM return code description

Description This function is called during the state transition from OP to SAFEOP(even if no output process data is available).

Prototype: **UINT16 APPL_GenerateMapping (UINT16 *pInputSize, UINT16 *pOutputSize)**

Parameter Pointer to two 16bit variables to store the process data size.
pInputSize : Input process data (Slave -> Master).
pOutputSize : Output process data (Master -> Slave).

Return See generic ESM return code description

Description This function is called when the transition from PREOP to SAFEOP is requested by the EtherCAT master. This function shall calculate the process data size in bytes The values are required to check the SyncManager settings and for the generic process data handling.

Prototype: **Void APPL_AckErrorInd(UINT16 stateTrans)**

Parameter stateTrans : Indicates the current state transition .

Return Void

Description This function is called when the master acknowledge and Error.

5.2.3 Process data handling

Prototype: **void APPL_InputMapping(UINT16 *pData)**

Parameter pData Pointer to the input process data.

Return Void

Description This function is called after the application call to map the input process data to the generic stack (The generic stack will copy the data to the SM buffer).

Prototype: **void APPL_OutputMapping(UINT16 *pData)**

Parameter pData Pointer to the output process data.

Return Void

Description This function is called before the application call to get the output process data.

6 Objects

To provide local memory to an EtherCAT master it should be handled as an object. An object is uniquely defined by the following characteristics:

- Local memory (6.1 Define local memory)
- Object Description (6.2 Object description)
- Object Name (6.3 Object name)
- Object Dictionary Entry Description (6.4 Object dictionary entry description)

The characteristics for all Objects are collected in one panel, the Object Dictionary. This Dictionary represents the Interface for the EtherCAT master. In this paragraph we classify the objects into three different Object Codes: VARIABLE, ARRAY and RECORD. The VARIABLE includes just one base data type as one object. The ARRAY is a collection of identical base data types as one object. The RECORD includes a collection of different base data types as one object.

In the next chapter the object characteristics are further described using Slave Stack Code examples.

6.1 Define local memory

The allocation of the local memory depends on the Object Code (VARIABLE, ARRAY or RECORD) which will be used.

In most cases for the Object Code VARIABLE it is sufficient to allocate memory by definition of a variable - if the desired object size is equal to a platform defined data type.

Example: Define local memory (Object Code VARIABLE)

```
UINT32 u32VarObject;
```

The other two Object Codes (ARRAY and RECORD) will be defined by structure. This contains an 8Bit variable as first member (Subindex0), which contains the highest subindex (last Object Entry). Note: in The Slave Stack Code the Subindex0 will always be defined as 16Bit variable due to alignment reasons!

Example: Define local memory (Object Code ARRAY)

```
typedef struct {
    UINT16    u16SubIndex0 ;
    UINT32    aEntries[4];
} _ARR_OBJ_DEF;

_ARR_OBJ_DEF ArrObj;
```

Example: Define local memory (Object Code RECORD)

```
typedef struct {
    UINT16    u16SubIndex0;
    UINT8     u8FirstEntry;
    UINT32    u32SecondEntry;
    INT16     i16ThirdEntry;
} _REC_OBJ_DEF;

_REC_OBJ_DEF RecObj;
```

6.2 Object description

The object description is defined in TSDOINFORMATIONDESC. The member variables are listed in

Table 2: TSDOINFORMATIONDESC member variables

Member	Data type	Description
DataType	unsigned 16 bit	Index of the base data type defined in [REF2].
BitLength	unsigned 16 bit	bit length of the object (entry)
ObjAccess	unsigned 16 bit	Bit 0: Read Access in Pre-Op Bit 1: Read Access in Safe-Op Bit 2: Read Access in Op Bit 3: Write Access in Pre-Op Bit 4: Write Access in Safe-Op Bit 5: Write Access in Op Bit 6: map able in RxPDO Bit 7: map able in TxPDO Bit 8: entry will be included in backup Bit 9: entry will be included in settings Bit 10: safe inputs Bit 11: safe outputs Bit 12: safe parameter

Example: Object description (Object Code VARIABLE)

```
TSDOINFORMATIONDESC VarObjectEntryDesc =
{DEFTYPE_UNSIGNED32, 0x20, (ACCESS_READ|OBJACCESS_TXPDOMAPPING)};
```

Example: Object description (Object Code ARRAY)

```
TSDOINFORMATIONDESC ArrObjEntryDesc[] = {
    {DEFTYPE_UNSIGNED8, 0x08, ACCESS_READ},
    {DEFTYPE_UNSIGNED32, 0x20, ACCESS_READ}};
```

Note: The object entry only describes the Subindex0 and one entry because all entries are equal.

Example: Object description (Object Code RECORD)

```
TSDOINFORMATIONDESC RecObjEntryDesc[] = {
    {DEFTYPE_UNSIGNED8, 0x08, ACCESS_READ},
    {DEFTYPE_UNSIGNED8, 0x08, ACCESS_WRITE_PREOP},
    {DEFTYPE_UNSIGNED32, 0x20, (ACCESS_WRITE|OBJACCESS_RXPDOMAPPING)},
    {DEFTYPE_INTEGER16, 0x10, ACCESS_WRITE}};
```

6.3 Object name

The Object Name is an ASCII coded name. The Object Codes VARIABLE and ARRAY are described only by one name including the escape sequence "\0".

If the Object Code is RECORD each Entry (except Subindex0) is described. Between two names the sequence "\000" is added. The escape sequence string "\000\377" is added.

Example: Object name (Object Code VARIABLE)

```
UCHAR VarObjName[] = "Test var obj";
```

Example: Object name (Object Code ARRAY)

```
UCHAR ArrObjName[] = "Array Obj";
```

Example: Object name (Object Code RECORD)

```
UCHAR RecObjName[] = "Record Obj\000First Entry\000Second Entry\000Third  
Entry\000\377";
```

6.4 Object dictionary entry description

The object dictionary entry description connects all object characteristics in one type "TOBJECT" (structure "OBJECT"). The member variables of OBJECT are listed in Table 3.

Table 3: "OBJECT" member variables

Member	Data type	Description
Prev Entry	struct OBJECT	Pointer to previous dictionary entry. Only available if the object dictionary entries are dynamic linked (STATIC_OBJECT_DIC = 0).
Next Entry	struct OBJECT	Pointer to next dictionary entry. Only available if the object dictionary entries are dynamic linked (STATIC_OBJECT_DIC = 0).
Index	unsigned 16 bit	Object index of the described object. The object value depends on the type of EtherCAT slave and object usage (6.5Index Ranges)
ObjDesc Data Type ObjFlags	TSDOINFOOBJDESC (32 bit) unsigned 16 bit unsigned 16 bit	Includes the data type index of the object. (defined in [2]) Bit 0-7: Max Subindex (value of subindex 0) Bit 8-15: Object Code (defined in [2])
pEntryDesc	TSDOINFOENTRYDESC *	Pointer to object description. Defined in "6.2Object description"
pName	unsigned char *	Pointer to object name. Defined in "6.3Object name"
pVarPtr	void *	Pointer to local memory. Defined in "6.1Define local memory"
Read		Pointer to Read Function. The prototype is listed below. This function will be called when an SDO upload is received. If this pointer is NULL the standard SDO upload function is executed. <u>Prototype:</u> UINT8 ReadFunction (UINT16 Index, UINT8 Subindex, UINT32 Size, UINT16 MBXMEM * pData, UINT8 bCompleteAccess)
Write		Pointer to Write Function. The prototype is listed below. This function will be called when an SDO download is received. If this pointer is NULL the standard SDO download function is executed. <u>Prototype:</u> UINT8 WriteFunction(UINT16 index, UINT8 subindex, UINT32 dataSize, UINT16 MBXMEM * pData, UINT8 bCompleteAccess)
NonVolatileOffset	unsigned 16 bit	determine offset within nonvolatile memory. This value is evaluated if the object should be stored(load) as backup parameter.

Example: Object dictionary entry description (Object Code VARIABLE)

```
TOBJECT VarObj_ODEntryDesc = {0x6000, {DEFTYPE_UNSIGNED32, 0 | (OBJCODE_VAR << 8)}, &VarObjectEntryDesc, VarObjName, &u32VarObject, NULL, NULL, 0x0000};
```

Example: Object dictionary entry description (Object Code ARRAY)

```
TOBJECT ArrObj_ODEntryDesc = {0x9000, {DEFTYPE_UNSIGNED32, 5 | (OBJCODE_ARR << 8)}, ArrObjEntryDesc, ArrObjName, &ArrObj, NULL, NULL, 0x0000 };
```

Example: Object dictionary entry description (Object Code RECORD)

```
TOBJECT RecObj_ODEntryDesc = {0x7000, {DEFTYPE_RECORD, 4 | (OBJCODE_REC << 8)}, RecObjEntryDesc, RecObjName, &RecObj, NULL, NULL, 0x0000 };
```

6.5 Index Ranges

The Object Index depends on the EtherCAT slave and the usage of the object. The Basic index ranges are listed in Table 4: Basic object index ranges.

Table 4: Basic object index ranges

Index Range	Description
0x0000 – 0x0FFF	Data Type Area
0x1000 – 0x1FFF	Communication Area
0x1600 – 0x19FF	RxPDO Mapping
0x1A00 – 0x1BFF	TxPDO Mapping
0x1C10 – 0x1C2F	Sync Manager PDO Assignment
0x1C30 – 0x1C4F	Sync Manager Parameters
0x2000 – 0x5FFF	Manufacturer specific Area
0x6000 – 0x6FFF	Input Area
0x7000 – 0x7FFF	Output Area
0x8000 – 0x8FFF	Configuration Area
0x9000 – 0x9FFF	Information Area
0xA000 – 0xAFFF	Diagnosis Area
0xB000 – 0xBFFF	Service Transfer Area
0xC000 – 0xEFFF	Reserved Area
0xF000h – 0xFFFF	Device Area

If the EtherCAT slave supports CiA402 drive profile the object range 0x6000 – 0xDFFF is subdivided according to [1]. The CiA402 objects used in the CiA402 sample listed in 8.1Objects.

The object indices used in the EL9800 Application are used according to the Modular device Profile (Figure 3: EL9800 Application object ranges).

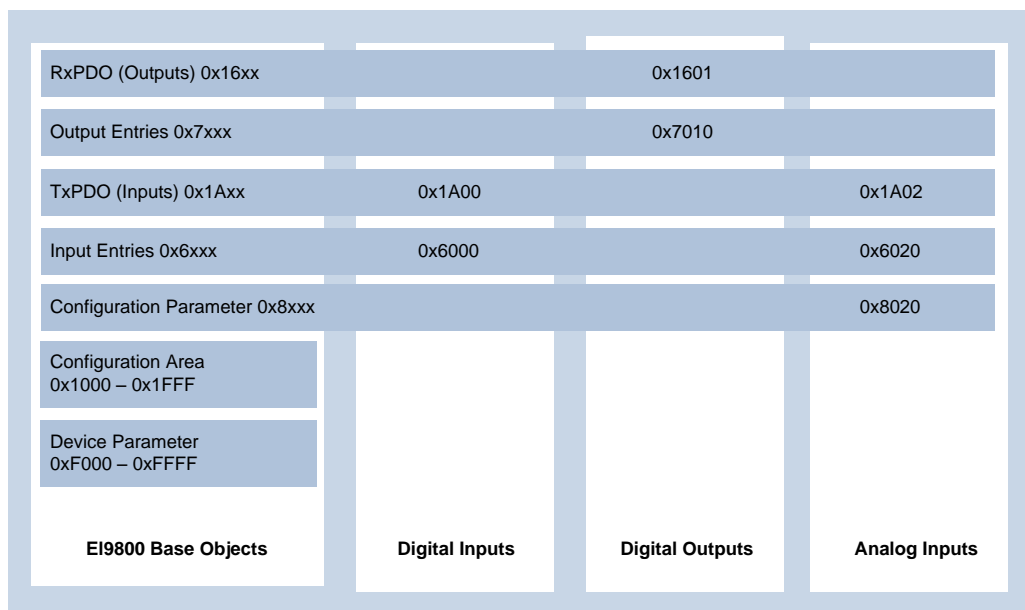


Figure 3: EL9800 Application object ranges

6.6 Implementation examples

6.6.1 Usage of Object Deftype ENUM

Each Enum Object definition shall be within the index range 0x800 – 0x0FFF. The content of an Enum definition is described in [2] .

For each enum Value a 4Byte unsigned integer Value and a Name is defined. The 4Byte unsigned integer is Byte wise Octed coded .

e.g: "\058\000\000\000EnumValueName"

"\058" = 5*8 + 8 = 48 = 0x30

=> "EnumValueName" = 0x00000030 = 48

Example: Define Object Deftype ENUM

```
CHAR sEnum0801_Value00[] = "\000\000\000\000Startup"; /* Value = 0x00, Text = Startup*/
CHAR sEnum0801_Value01[] = "\001\000\000\000Runnig"; /* Value = 0x01, Text = Runnig*/
CHAR sEnum0801_Value02[] = "\012\000\000\000End"; /* Value = 0xA, Text = End*/
CHAR *apEnum0801[] = { sEnum0801_Value00, sEnum0801_Value01, sEnum0801_Value02};

OBJCONST TSDOINFOENTRYDESC OBJMEM asEntryDesc0x0801[] =
    {{DEFTYPE_UNSIGNED8, 8, ACCESS_READ | OBJACCESS_NOPDOMAPPING},
     {DEFTYPE_OCTETSTRING, 8*SIZEOF(sEnum0801_Value00), ACCESS_READ | OBJACCESS_NOPDOMAPPING},
     {DEFTYPE_OCTETSTRING, 8*SIZEOF(sEnum0801_Value01), ACCESS_READ | OBJACCESS_NOPDOMAPPING},
     {DEFTYPE_OCTETSTRING, 8*SIZEOF(sEnum0801_Value02), ACCESS_READ | OBJACCESS_NOPDOMAPPING}};
```

Example: Define Object dictionary entry: ENUM Object

```
{NULL, NULL, 0x0801, {DEFTYPE_ENUM, 0x03 | (OBJCODE_REC << 8)}, asEntryDesc0x0801, 0, apEnum0801 },
```

Example: Define New Object (using ENUM)

```
OBJCONST TSDOINFOENTRYDESC OBJMEM sEntryDesc0x2001 = {0x801, 0x20, ACCESS_READ };
OBJCONST UCHAR OBJMEM aName0x2001[] = "MySampleObject";
UINT32 u32MyObject = 0xA;
```

Example: Object dictionary entry description New Object (using ENUM)

```
TOBJECT MyObject_ODEntryDesc = { 0x2001, {0x801, 0 | (OBJCODE_VAR << 8)}, &sEntryDesc0x2001, aName0x2001, & u32MyObject, NULL, NULL, 0x0000 },
```

7 Mailbox

7.1 FoE (File Transfer over EtherCAT)

FoE can be used to download and upload a file to an EtherCAT device. The protocol is similar to TFTP service.

7.1.1 Testing FoE

The Slave Stack Code supports FoE not by default but it can be activated and tested:

- 1.) Set the defines in "ecat_def.h"
 - a. FOE_SUPPORTED 1
 - b. FOE_SAVE_FILES 1
(FoE is also available if this define is not set but the downloaded file will not be saved)
- 2.) Build a binary file (*.hex) (see [3])
- 3.) Write binary to the PIC controller of the Evaluation Kit (see Application Note EL9800 [3]).
- 4.) Check if FoE flag in ESI file
 - a. Open ESI file (..\SlaveStackCode\VXiXX\esi\SlaveStackCode.xml)
 - b. Open corresponding entry
 - i. Board 4a (new board): EL9800-SPI-PIC24
 - ii. Board 2 (former board): EL9800-SPI-PIC18
 - iii. If CiA 402 example is used: EL9800-CiA402
 - c. Check if the element EtherCATInfo : Descriptions : Devices : Device : Mailbox : FoE is available. It has only to be present. No content is necessary.
- 5.) If the ESI file was edited replace the original ESI file from TwinCAT (..\Io\Ethercat\SlaveStackCode.xml) with the edited one.
- 6.) Rewrite SII if ESI file was edited (see[3])
- 7.) Restart device and restart TwinCAT
- 8.) Scan Network
- 9.) Select device (Evaluation Kit with FoE activated)
- 10.) Select Tab-"Online" and read or write a file to the slave device by pressing the corresponding buttons in the File Access over EtherCAT – group box.

7.2 EoE (Ethernet over EtherCAT)

EoE is used to send Ethernet telegrams to EtherCAT devices (supporting a Ethernet stack). EoE is used for Ethernet communication i.e. a device supports a web server that can be accessed via browser.

7.2.1 Implementation

The EoE stack is implemented in the files *ecateoe.** (basic EoE handling) and *eoappl.** (EoE application handling). To support EoE the switch "EOE_SUPPORTED" need to be set to 1 and the EoE flag need to be set in the ESI (REF.[4]). Depending on the switch "STATIC_ETHERNET_BUFFER" dynamic memory is allocated for Ethernet frame handling or a fixed memory is used (1514 Bytes). By default the stack only handles ICMP and ARP frames.

7.2.1.1 Sending EoE datagrams

To send EoE datagram from the EtherCAT slave to the EtherCAT master the function "EOE_SendFrameReq()" shall be called. The program flow is shown in

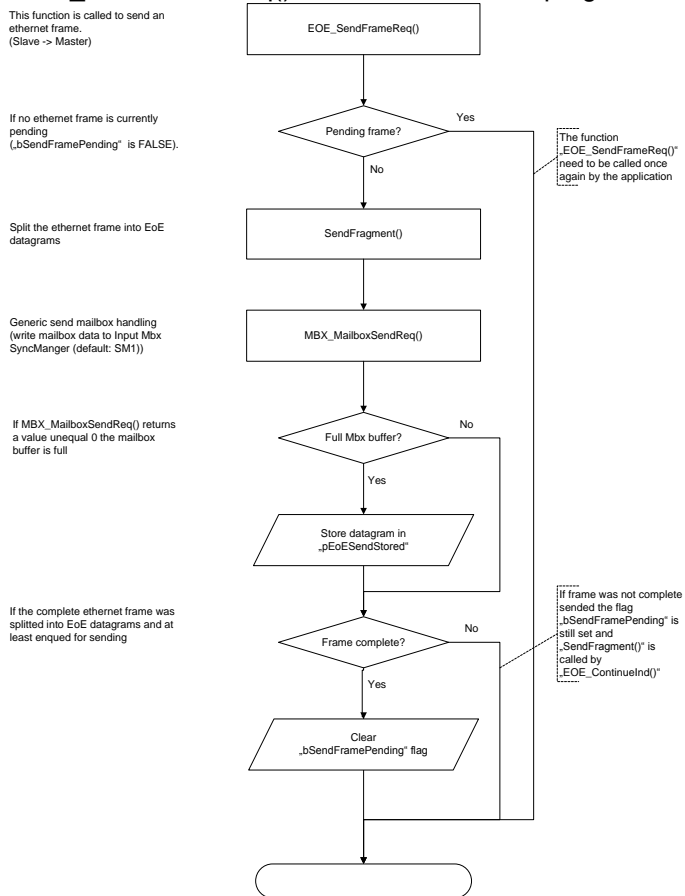


Figure 4.

This function is called to send an ethernet frame.
(Slave -> Master)

If no ethernet frame is currently pending
(„bSendFramePending“ is FALSE).

Split the ethernet frame into EoE datagrams

Generic send mailbox handling
(write mailbox data to Input Mbx SyncManger (default: SM1))

If MBX_MailboxSendReq() returns a value unequal 0 the mailbox buffer is full

If the complete ethernet frame was splitted into EoE datagrams and at least enqueued for sending

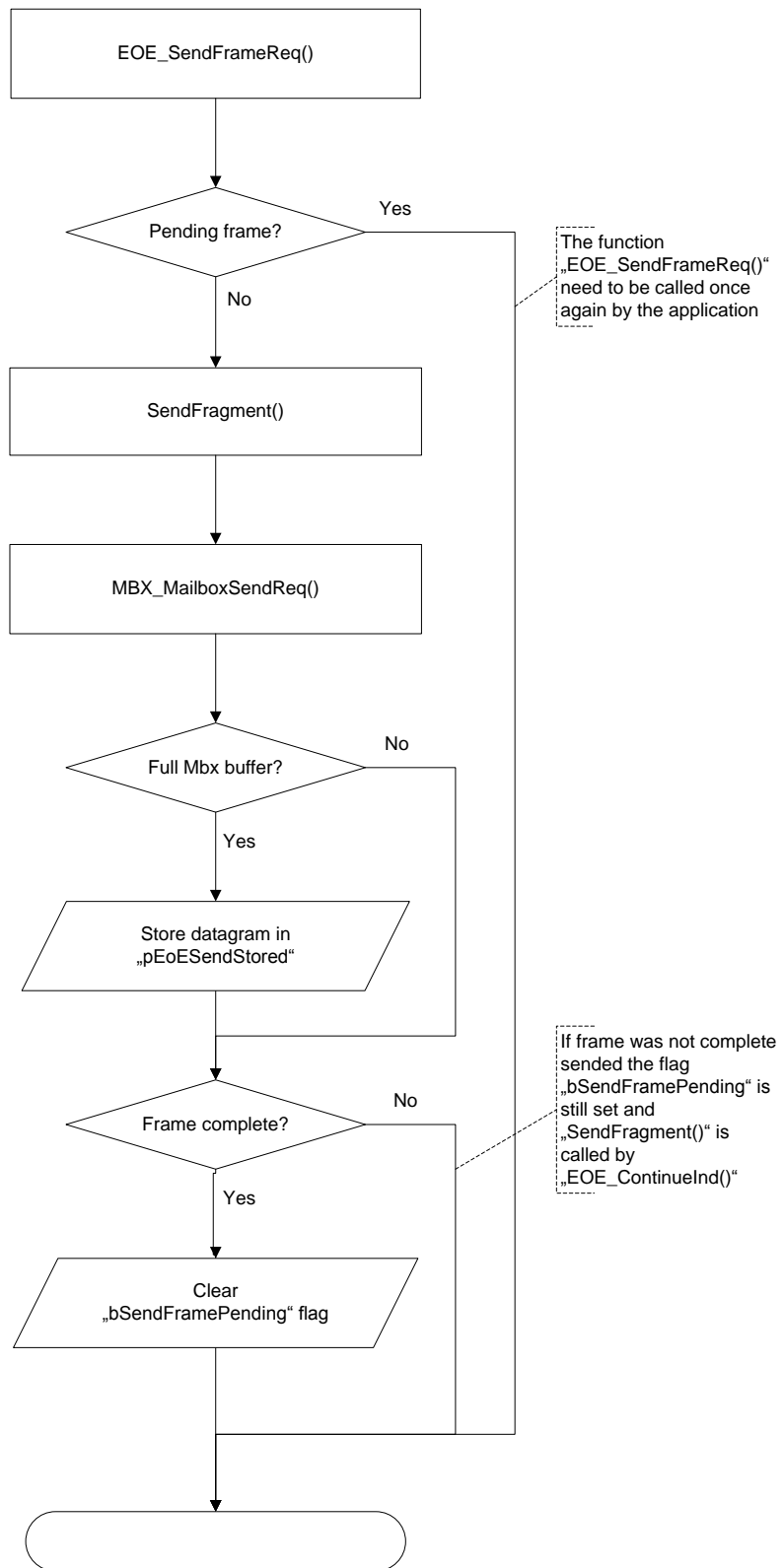


Figure 4: Send EoE datagram

7.2.1.2 Receiving EoE datagrams

Received EoE datagrams are handled by the function “EOE_ServiceInd()” which is called from the generic mailbox stack. The program flow is shown in Figure 5.

This function is called from the generic mailbox stack if a EoE datagram was received

Checks if an EoE Init Request was received

Received MAC („aMacAdd“) and IP („alpAdd“) address are stored.

Checks if ethernet fragment was received via EoE. All other EoE services are not supported by the SSC

Depending on the switch „STATIC_ETHERNET_BUFFER“ dynamic memory is allocated or the frame is stored in „aEthernetReceiveBuffer“

If the complete ethernet frame was received the EoE application shall be triggered

By default only ICMP and ARP frames are handled by this function

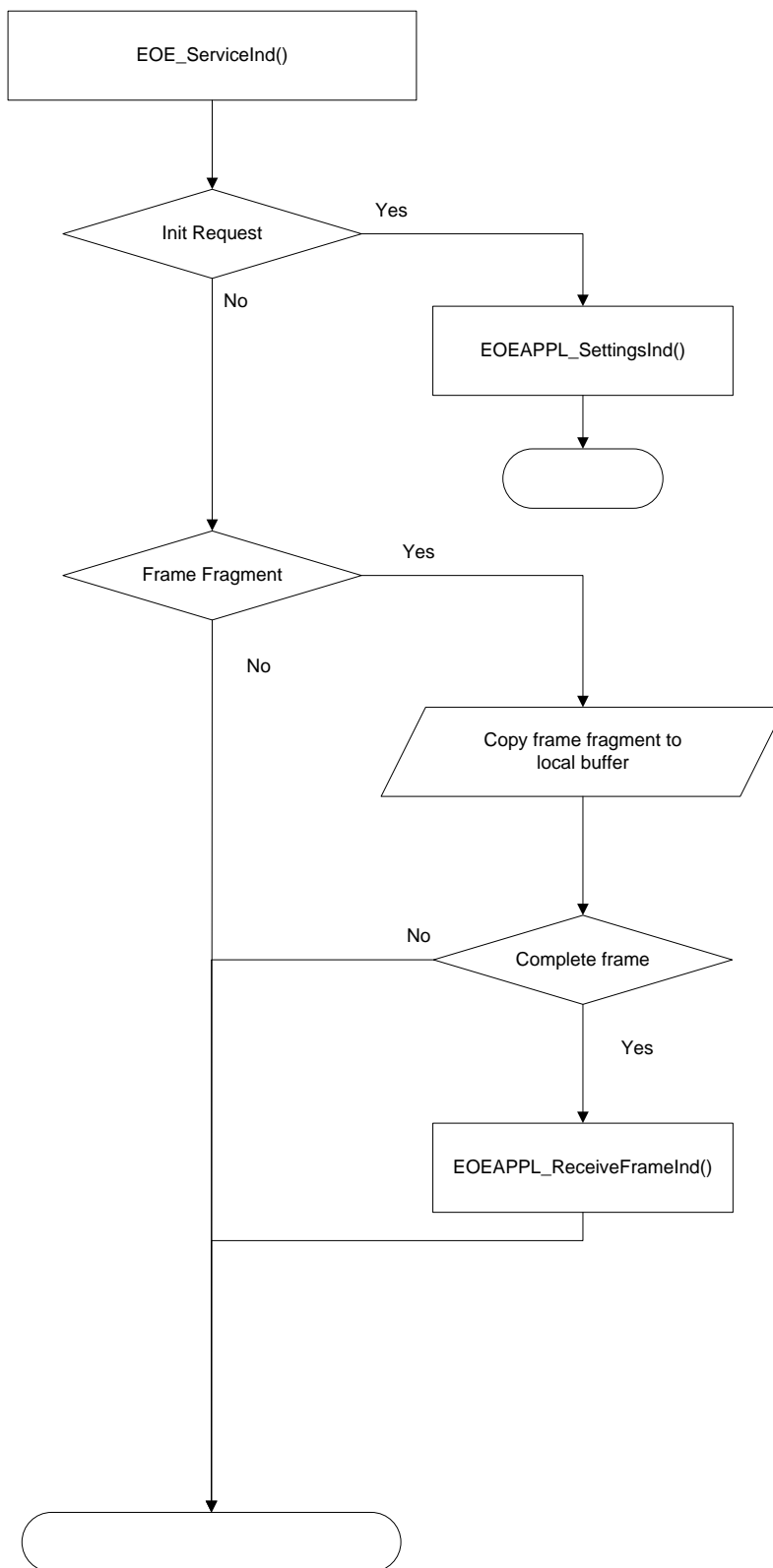


Figure 5: Receive EoE datagram

7.2.2 EoE Examples

The Sample code has a simple ping service integrated that answers to a ping request.

- 1.) Set the define EOE_SUPPORTED 1 (in ecat_def.h)
- 2.) Build a binary file (*.hex) (see [3]).

- 3.) Write binary to the PIC controller of the Evaluation Kit (see Application Note EL9800 [3]).
- 4.) Change EoE flag in ESI file
 - a. Open ESI file (..\SlaveStackCode\VXiXX\esi\SlaveStackCode.xml)
 - b. Open corresponding entry
 - i. Board 4a (new board): EL9800-SPI-PIC24
 - ii. Board 2 (former board): EL9800-SPI-PIC18
 - iii. If CiA 402 example is used: EL9800-CiA402
 - c. Enter the element EtherCATInfo : Descriptions : Devices : Device : Mailbox : EoE
Only the element has to be present.
- 5.) Replace the original ESI file from TwinCAT (..\Io\Ethercat\SlaveStackCode.xml) with the edited one.
- 6.) Rewrite SII (see [3]).

7.2.2.1 EoE Example 1

The prerequisite for this example are the steps described in the introduction of chapter 7.2.2 EoE Examples.

The example describes how to ping an EtherCAT slave device from a master platform (Figure 6: EoE Example 1 (Schema)).

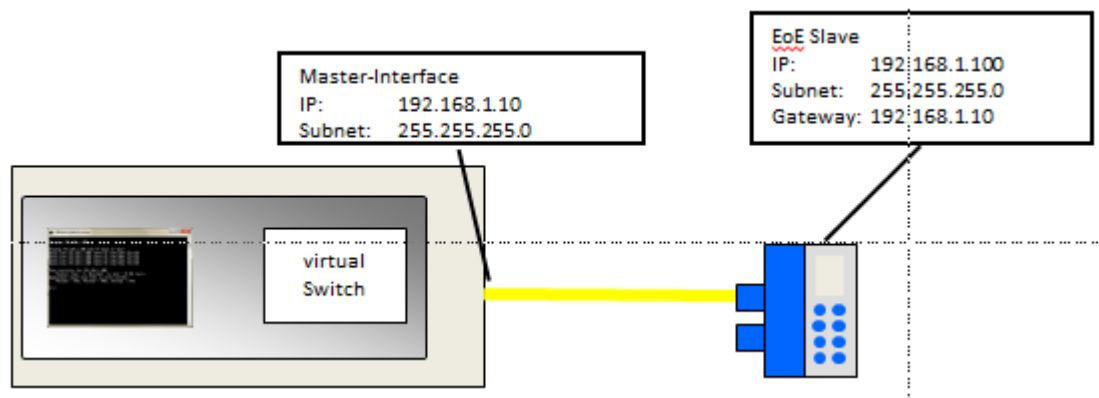


Figure 6: EoE Example 1 (Schema)

- 1.) Restart device and restart TwinCAT
- 2.) Configure Network Card NIC
 - a. Open network adapter setting
 - b. Open the settings of the Network-Card that is used for EtherCAT (!)
 - i. Set IP-Address of the card to the value you want to use, e.g.:
 - ii. IP-Address: 192.168.1.10
 - iii. Subnet Mask: 255.255.255.0
 - c. Leave all other fields blank (DNS, WINS, Gateway)

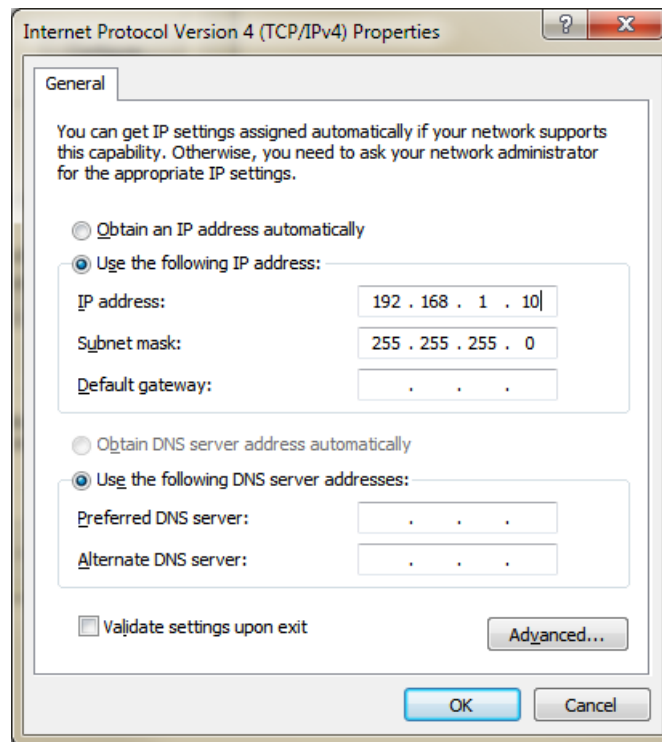


Figure 7: Network card settings

- 3.) Save settings
 - a. Configure device
 - b. Open TwinCAT
 - c. Scan Network
 - d. Select device (Evaluation Kit with EoE activated)
 - e. Select EtherCAT tab and [Advanced Settings]

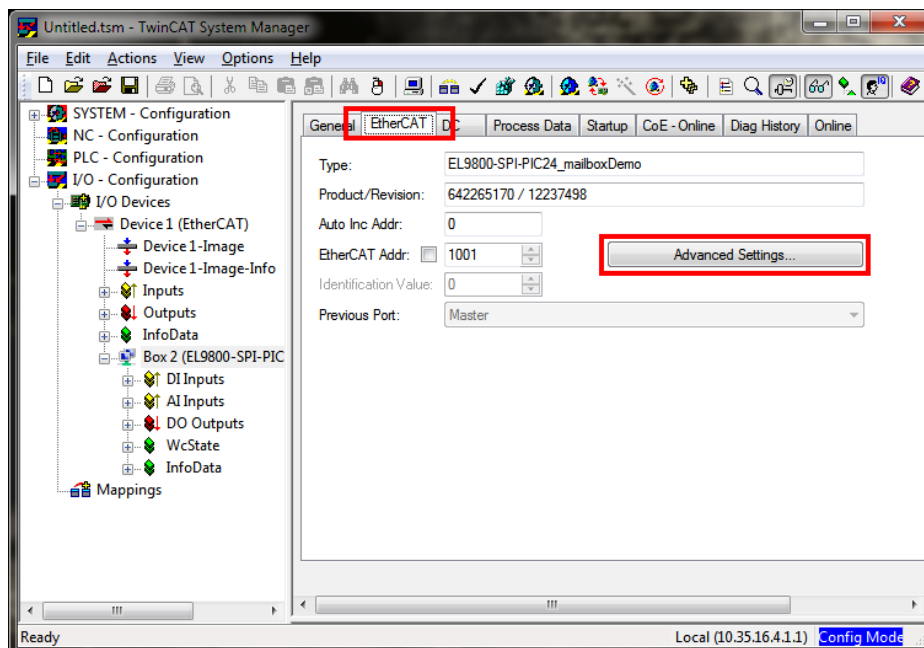


Figure 8: Access EtherCAT Slave Settings

- f. Configure a IP address in the same subnet
- g. Set the IP address of the NIC as gateway

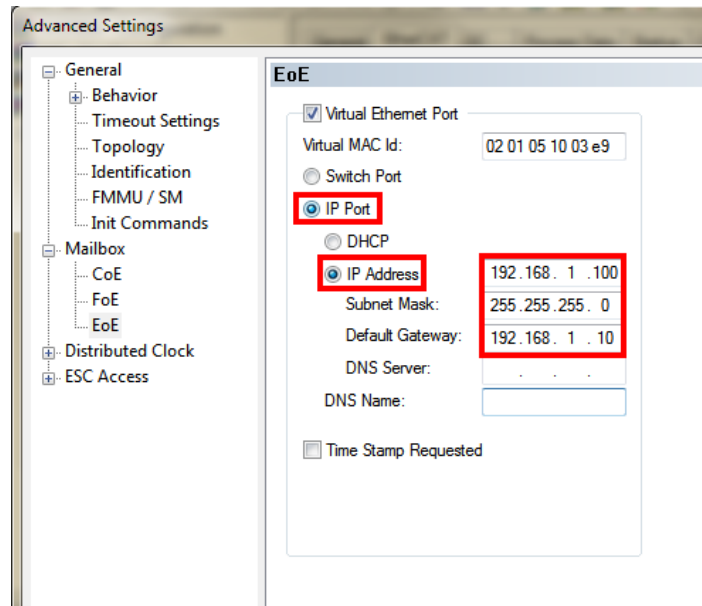


Figure 9: EoE EtherCAT Slave Settings

- 4.) Set network at least to PRE-OP (mailbox communication needed)
- 5.) Open a program supporting PING service
- 6.) Ping device

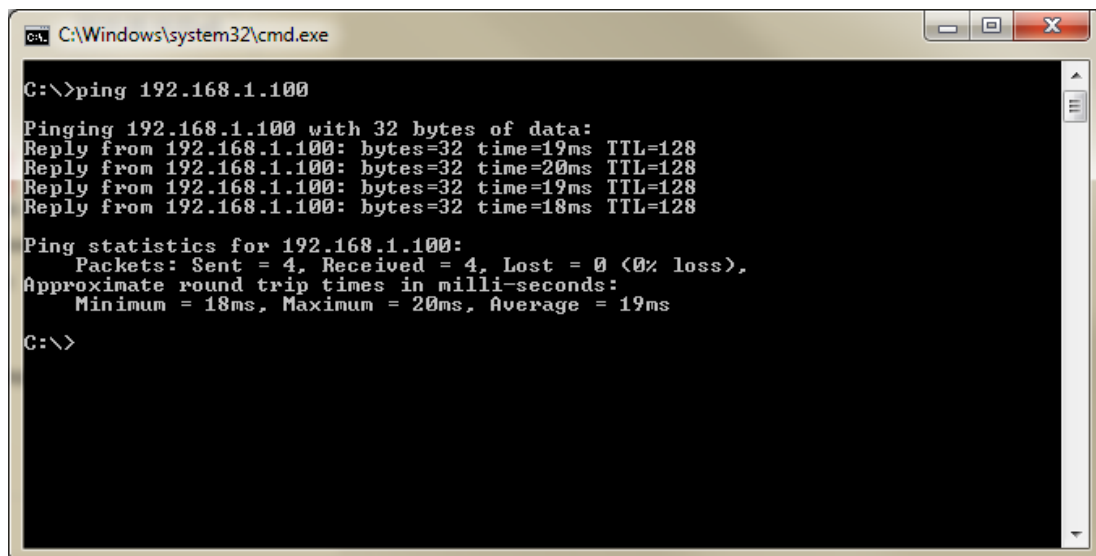


Figure 10: Ping Command Window

7.2.2.2 EoE Example 2

The prerequisite for this example are the steps described in the introduction of chapter 7.2.2 EoE Examples. The example describes how to ping an EtherCAT slave device from a remote PC (Figure 11: EoE Example 2 (Schema)).

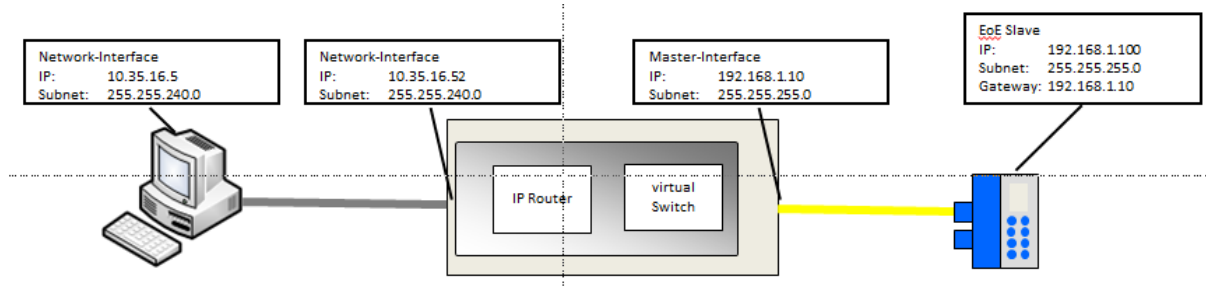


Figure 11: EoE Example 2 (Schema)

Steps 1 to 3 are equal to EoE Example 1 (chapter 7.2.2.1).

- 4.) Enable IP Routing on the EtherCAT Master platform. The following steps depend on the operating system.
 - a. Windows XP
 - i. Open the Advanced EtherCAT settings of the Master interface and select "IP Enable Router"().

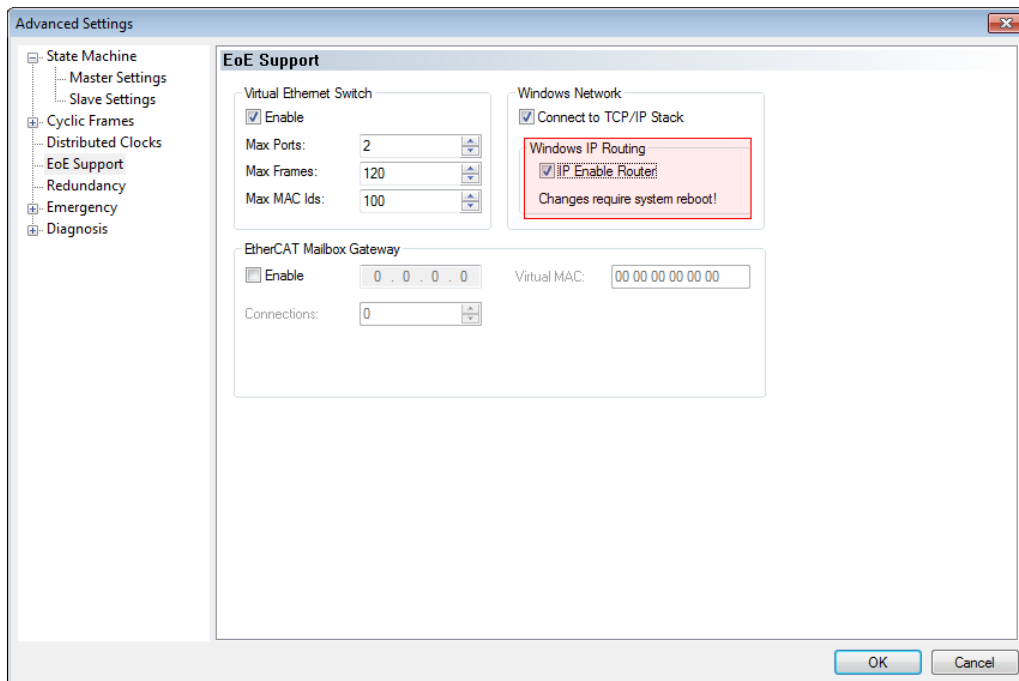


Figure 12: Enable IP Routing WinXP

- b. Windows CE (CX platform)
 - i. Open CX Configuration Tool and enable "IP Routing".

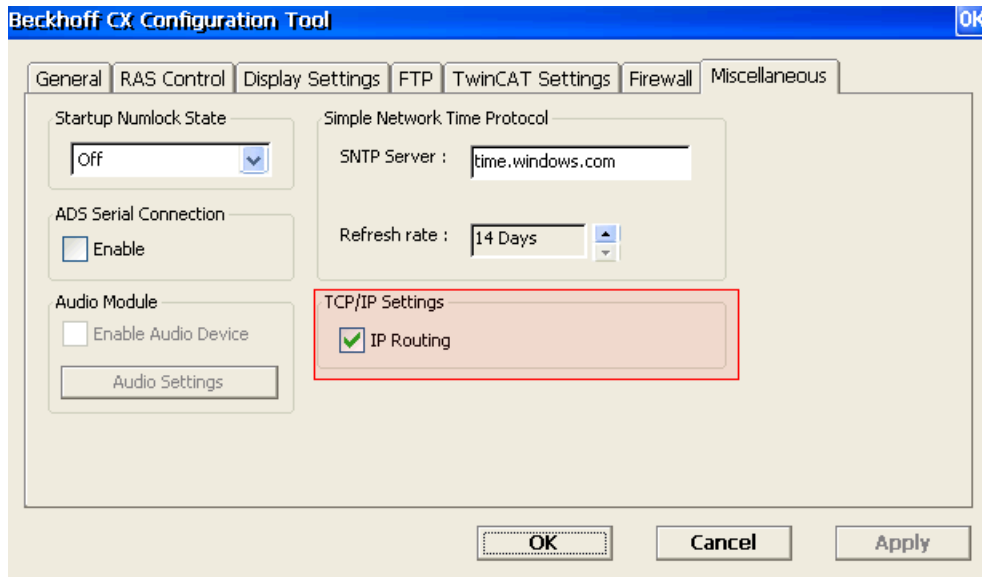


Figure 13: Enable IP Routing WinCE

- 5.) Restart the PC
- 6.) Add Route on the external PC
 - a. Command: `route ADD 192.168.1.0 MASK 255.255.255.0 10.35.16.52`
- 7.) Ping slave device

8 CiA402 drive profile

Since version 4.30 the Slave Stack Code contains a sample implementation of the CiA402 drive profile as described in [1]. This implementation provides the interface between the motion controller application and communication layer.

Following features are supported:

- CiA402 objects (see chapter 8.1 Objects)
- CiA402 state machine (see chapter 8.2 State machine)
- This implementation supports cyclic synchronous position (csp) and cyclic synchronous velocity (csv) operation modes.

CiA402 specific files:

`cia402appl.c` : CiA402 drive profile implementation

`cia402appl.h` : Drive profile specific objects, definitions and axes structures

All motion controller related values are encapsulated in structure `TCiA402Axis` (file: `cia402appl.h`). The configuration parameters and error codes are directly mapped to the corresponding objects. The process data objects are updated in the input/output mapping functions (file: `ecatappl.c`). Currently the sample supports maximum of two axes. The axes are initialized in the EtherCAT state change from PREOP to SAFEOP.

The motion controller is a simple integration, which just copies the target values to the actual values (see chapter 8.3 Operation modes).

8.1 Objects

All CiA402 specific objects are defined in file `cia402appl.h`.

All mandatory and some optional object are defined in this sample implementation. Table 5 contains a list of all defined objects. The object variables are located in the structure `CiA402Objects`.

Table 5: Object definitions in file cia402app1.h

Index	Object name	Variable in source code	Comment/Description
0x1600	Rx PDOs	sRxPDOMap0	includes all objects required for dynamic change between csv/csp
0x1601	Rx PDOs	sRxPDOMap1	includes objects required for csp mode of operation
0x1602	Rx PDOs	sRxPDOMap2	includes objects required for csv mode of operation
0x1A00	Tx PDOs	sTxPDOMap0	includes all objects required for dynamic change between csv/csp
0x1A01	Tx PDOs	sTxPDOMap1	includes objects required for csp mode of operation
0x1A02	Tx PDOs	sTxPDOMap2	includes objects required for csv mode of operation
0x1C12	SyncManger 2 PDO assign (Rx PDOs)	sRxPDOassign	this object is written in change state from PREOP to SAFEOP; the configuration depends on the number of axes (not include in CiA402Objects)
0x1C13	SyncManger 3 PDO assign (Tx PDOs)	sTxPDOassign	equal to 0x1C12 (not include in CiA402Objects)
0x603F	Error Code	objErrorCode	this value shall be set if an error in the PDS occurs
0x6040	Controlword	objControlWord	object for the output commands from the master
0x6041	Status word	objStatusWord	current axis status
0x605A	Quick stop option code	objQuickStopOptionCode	predefined ramp if an quick stop shall be performed
0x605B	Shutdown option code	objShutdownOptionCode	predefined action in state transition 8
0x605C	Disable operation option code	objDisableOperationOptionCode	predefined action in state transition 5
0x605E	Fault reaction option code	objFaultReactionCode	predefined action in state "Fault reaction active"
0x6060	Modes of operation	objModesOfOperation	requested operation mode
0x6061	Modes of operation display	objModesOfOperationDisplay	current operation mode
0x6064	Position actual value	objPositionActualValue	current position value (delivered by encoder)
0x606C	Velocity actual value	objVelocityActualValue	velocity feedback
0x6077	Torque actual value	objTorqueActualValue	currently not used (only for completion)
0x607A	Target position	objTargetPosition	requested Position value (set in csp mode)
0x607D	Software position limit	objSoftwarePositionLimit	includes the minimum and maximum actual position limit
0x6085	Quick stop declaration	objQuickStopDeclaration	predefined action in state "Quick stop active"
0x60C2	Interpolation time period	objInterpolationTimePeriod	
0x60FF	Target velocity	objTargetVelocity	target velocity requested by the master
0x6502	Supported drive modes	objSupportedDriveModes	list of all supported operation modes

The objects from 0x6000 to 0x67FF are incremented with 0x800 for each axis (Index + #Axis **Error! bookmark not defined.***0x800).

8.2 State machine

Figure 14 shows the state machine described in [1]. State changes are requested by setting 0x6040 (Controlword) or by a local event (if an error occurs). If the device is in state OP the transitions 0, 1 and 2 are skipped. The option codes next to the transition lines indicate that a specific action which shall be performed in of one of these state changes.

All handled state transitions including the required Controlword, resulting state and corresponding functions are listed in Table 6: State machine. The bits 0,1,2,3 and 7 of the control word are taken into consideration in this sample implementation. Drive functions e.g. "Break applied" or "Axis function enabled" need to be activated or deactivated corresponding to the current state. In this sample these functions are handled by Boolean variables.

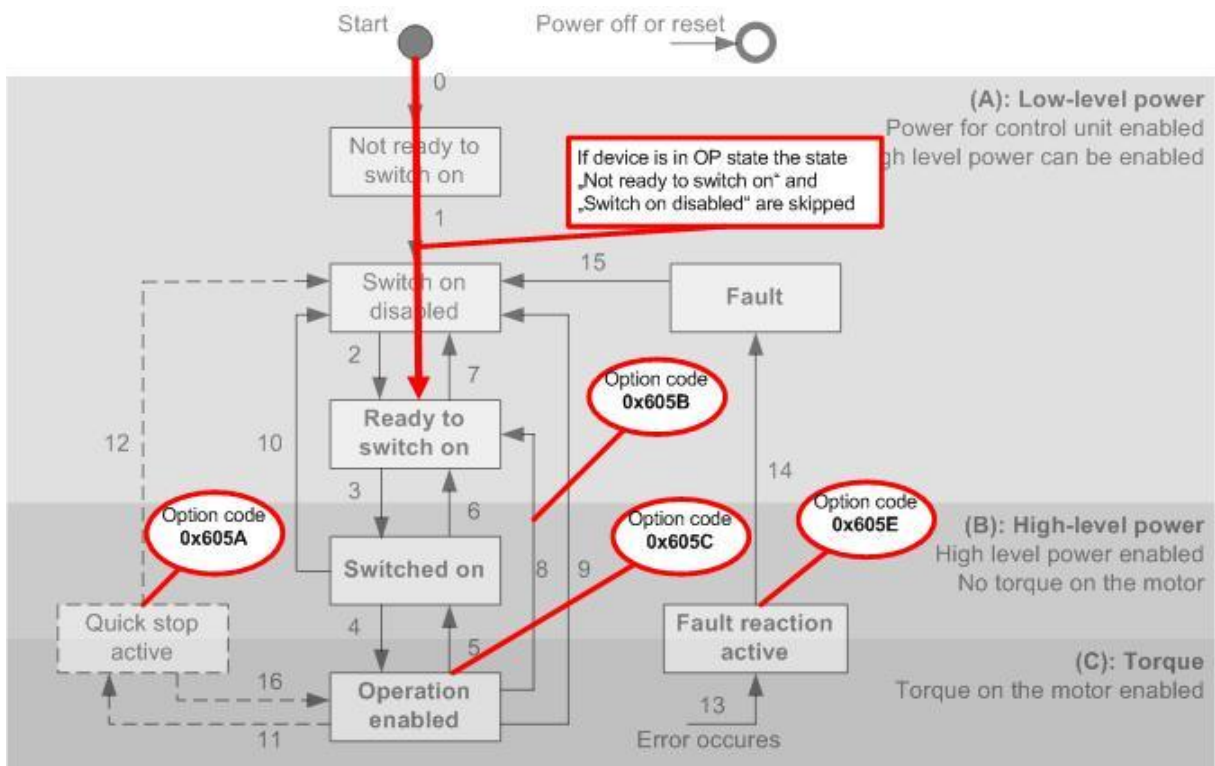


Figure 14: CiA402 state transitions and option codes

Table 6: State machine

Transition	Control word bits					Resulting state	Variable (Function)				
	Bit 7 (fault reset)	Bit 3 (enable operation)	Bit 2 (quick stop)	Bit 1 (enable voltage)	Bit 0 (switch on)		bBrakeApplied	bLowLevelPowerApplied	bHighLevelPowerApplied	bAxisFunctionEnabled	bConfigurationAllowed
3	0	x	1	1	1	Switched on	true	true	true	false	true
4	0	1	1	1	1	Operation enabled	false	true	true	true	false
5	0	0	1	1	1	Switch on	true	true	true	false	true
6	0	x	1	1	0	Ready to switch on	true	true	false	false	true
7	0	x	x	0	x	Switch on disabled	true	true	false	false	true
	0	x	0	1	x						
8	0	x	1	1	0	Ready to switch on	true	true	false	false	true
9	0	x	x	0	x	Switch on disabled	true	true	false	false	true
10	0	x	x	0	x	Switch on disabled	true	true	false	false	true
	0	x	0	1	x						
11	0	x	0	1	x	Quick stop active	false	true	true	true	false
12	0	x	x	0	x	Switch on disabled	true	true	false	false	true
13	Triggered by application					Fault reaction active	false	true	true	true	false
14	Transition if option code 0x605E is finished					Fault	true	true	false	false	true
15	1	x	x	x	x	Switch on disabled	true	true	false	false	true
16	After quick stop always goto "Switch on disabled"					(Operation enabled)	false	true	true	true	false

The transition number in Table 5 referring to the transition number in Figure 14.[1]

8.3 Operation modes

In general this sample supports the csv and csp mode of operation. Each axis can be configured as csv, csp or combined controller via modules (Figure 15). In last case the mode of operation can be switch dynamically. For this reason all objects required for motion control are mapped to PDOs. In the current TwinCAT Version (2.11 build 1539) the NC task doesn't provide a variable for the objects 0x6060 (Mode of operation) and 0x6061 (Mode of operation display), so these objects values need to be directly provided by the PLC.

The motion controller function (`CiA402_DummyMotionControl()`) just copies the target velocity values to the actual velocity. The actual position is calculated by the actual velocity and the motion controller cycle time. If the device is in SM Sync mode the cycle time is calculated by an internal timer within the first application cycle. In DC Sync mode the cycle time is set to Sync0 cycle value.

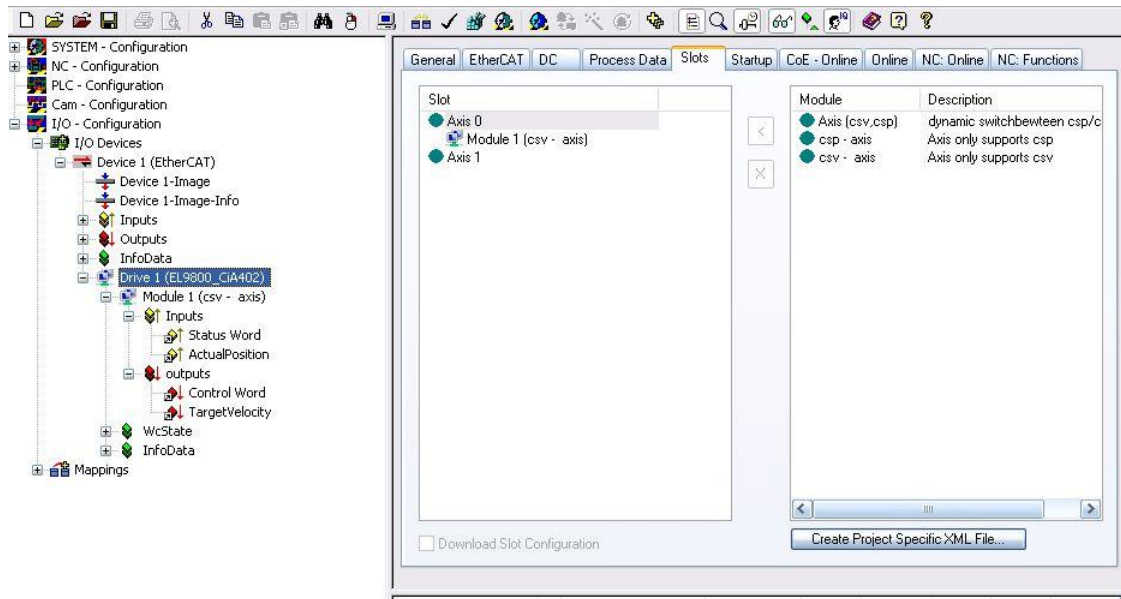


Figure 15: Axis configuration



NOTE: This sample doesn't provide a target value overflow control!

8.4 TwinCAT setup

This chapter describes the setup of motion control loop over EtherCAT. It is based on TwinCAT Version V2.11 build 1539. At least the TwinCAT level NC is required. The position control is located on the EtherCAT master so in this case only the "target velocity" and "actual position" need to be linked to the NC task. The mode of operation shall be set to cyclic synchronous velocity mode (csv). For the corresponding objects 0x6061 (Mode of operation display) and 0x6060 (Mode of operation) is no NC axis variable reserved. So these drive variables should be mapped direct to the PLC application.

For testing purposes 0x6060 (Mode of operation) could be set manually at each EtherCAT master restart.

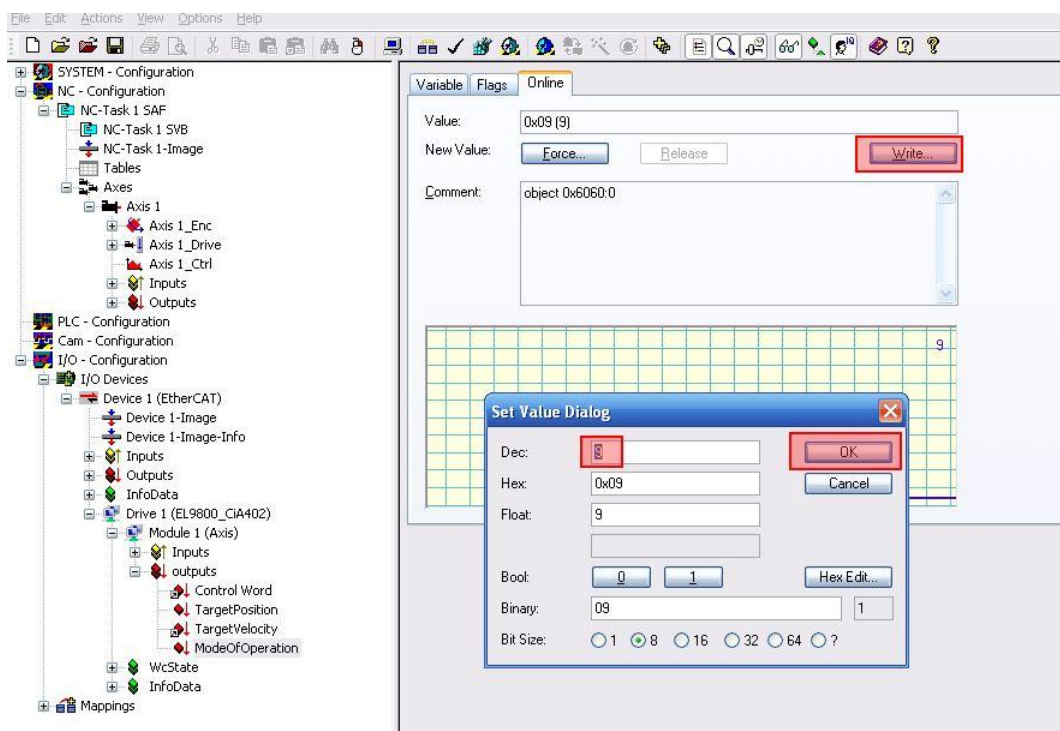


Figure 16: Set device variable without PLC link

8.4.1 Automatic network setup

The TwinCAT System Manager provides a comfortable master setup. Open a new System Manager configuration and scan the network for new devices (Figure 17).

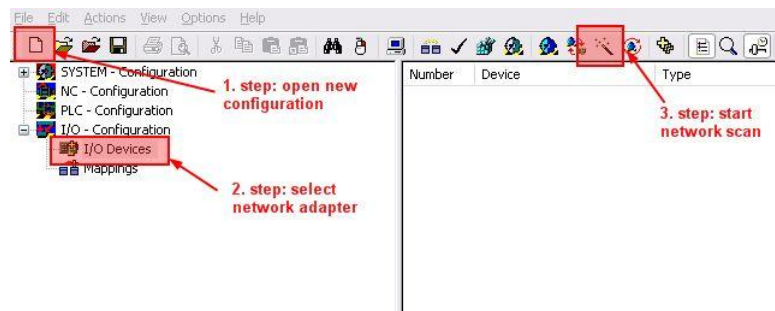


Figure 17: Scan for new EtherCAT devices

After the network scan is complete a message box appears with a notification that an EtherCAT drive was found. If this message is acknowledged with “Yes” the System Manager will automatically create an NC task with the correct process data mapping.

8.4.2 Manual network setup

First add a NC task including a CiA402 Axis to the System configuration. Add NC task ->Add continuous Axis -> set Axis type to “CANopen DS402” (see Figure 18).

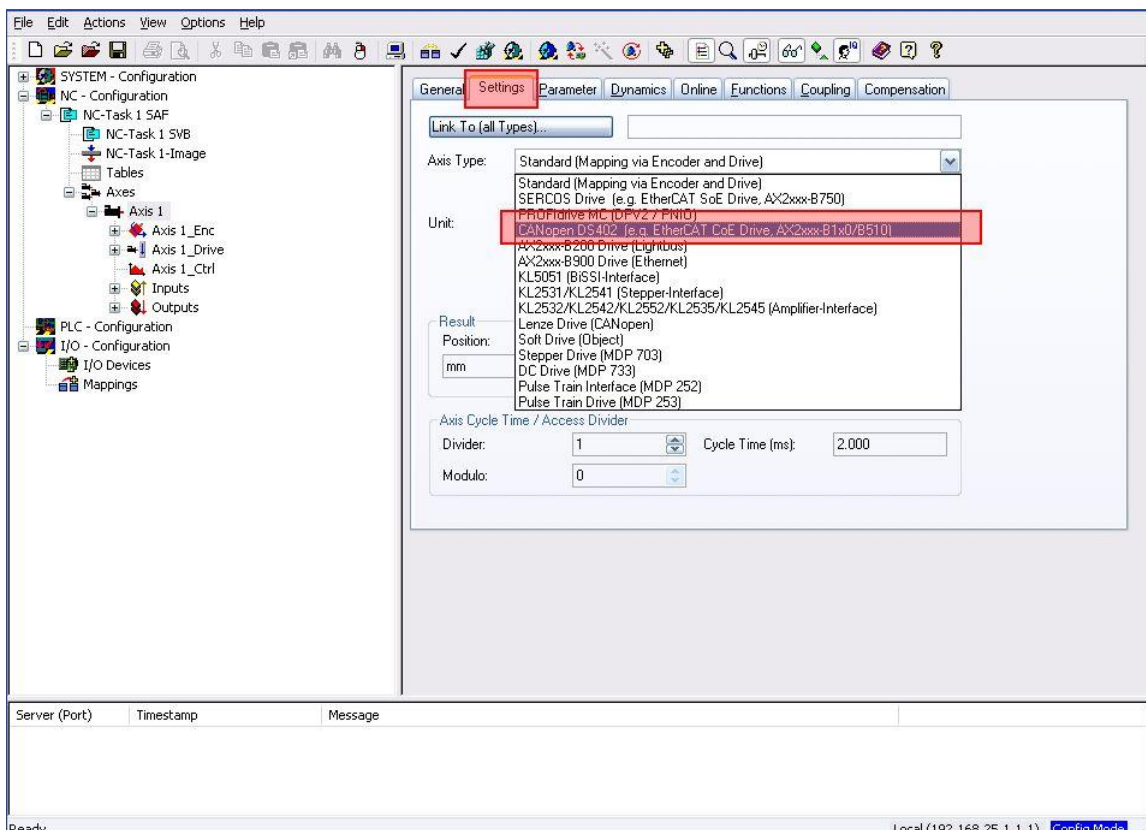


Figure 18: CiA402 axis setup

Add a new EtherCAT device and append the device “EL9800 (Vx.xx/CiA402) (PIC24, SPI, ET1100)” (Figure 19).

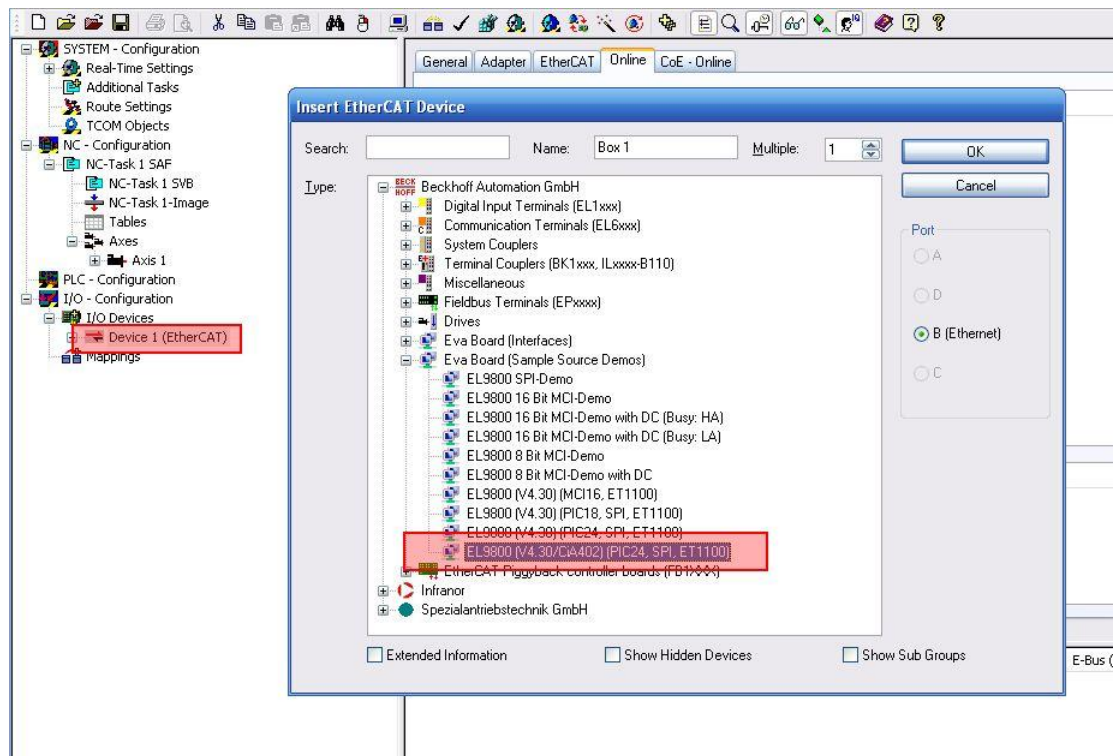


Figure 19: Add CiA402 device

Now the device variables (objects) need to be linked to the axis variables as shown in Table 7.

Table 7: Linking of device and NC variables

object index	type	variable name	
		device	NC axis
0x6040	output	Controlword	“Drive_Out” -> “nCtrl1” + “nCtrl2”
0x60FF	output	Target velocity	“Drive_Out” -> “nOutData2”
0x6041	input	Statusword	“Drive_In” -> “nStatus1” + “nStatus2”
0x6064	input	Actual position	“Enc_In” -> “nInData1”

For the Statusword and Controlword continuous process data mapping is required. This can be performed in the linking window (eg. Statusword link window Figure 20). Enable “All Types”, “Continuous” and select the desired variables. TwinCAT will map “nStatus1” to the low byte and “nStatus2” to the high byte of the Statusword.

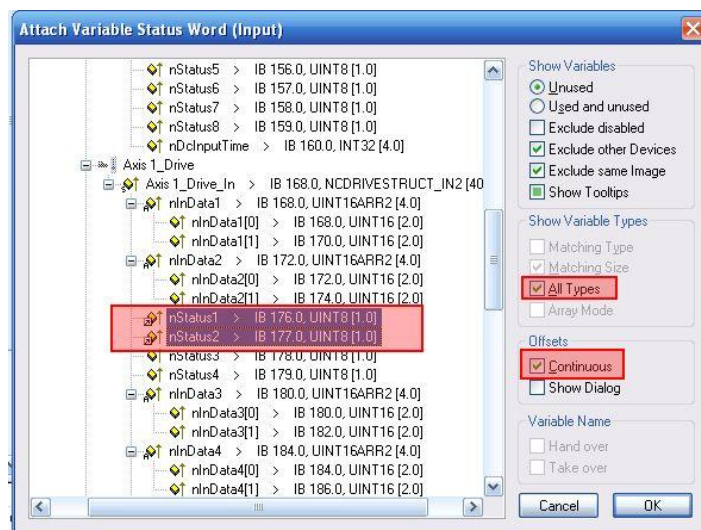


Figure 20: Link multiple variables

8.4.3 NC parameter setup

It is required to setup the encoder and velocity output scaling in the NC-task of the EtherCAT master according to the drive parameters. The sample implementation not supports user defined factor group objects so the default units are used.

- Position unit: inc
- velocity unit: inc/s
- Encoder resolution: 2¹⁶ inc/rev

If a non predefined drive is used two basic information are required, the **target velocity value for 1rev/min** (=> target velocity resolution) and the **encoder resolution**.

The encoder resolution defined as 2¹⁶ inc/rev and if 1 rev is equal to 1mm the encoder scaling factor [mm/inc] is 0.0000152588 (Figure 21).

Encoder scaling factor formula: $enc.scaling = (mm/rev)/encoder\ resolution$

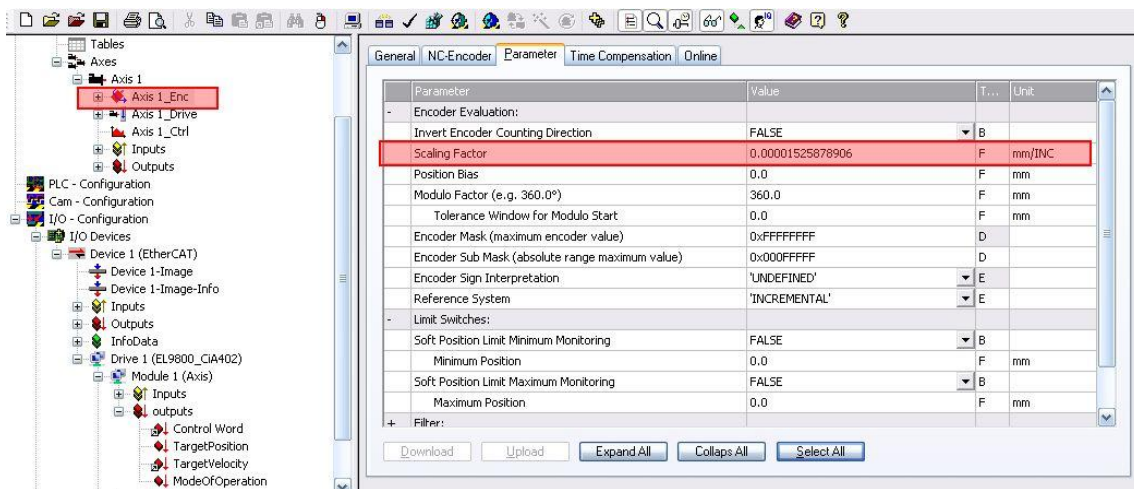


Figure 21: Encoder scaling

The velocity output scaling is calculated with the following formula:
 $velo.scaling = (2^{20} / encoder\ resolution) * (velo\ resolution / 139,81)$

The velocity resolution [inc/(1rev/min)] is the numerical increment if 1 rev/min is desired. In this case the ratio is 1 so the velocity scaling factor is 0.114441027(Figure 22).

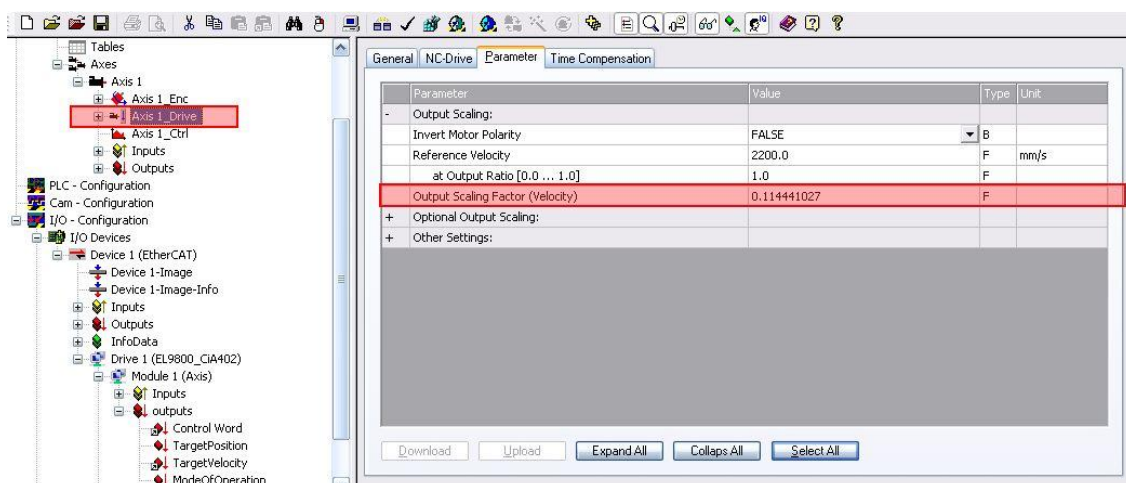


Figure 22: Velocity scaling

9 TestApplication

The test application is a specific slave stack which provides most of the specified EtherCAT slave features and also a mechanism to generate a slave behavior that which is not conform to the standard. This may be used to check master behavior with incorrect slave behavior.

This chapter is dealing with the possible configurations. The possible misbehaviors are organized within CoE objects in the index range from 0x2000 to 0x2FFD (Table 8 and Table 9).

Table 8: Test Object

Attribute	Value
Index	0x2000 to 0x2FFD
Name	Test object
Object Code	RECORD
Max SubIndex	1-255

Table 9: Test Object Entry

Sub-Index	Description	Data Type	Access	PDO Mapping	Description / Default value	
n (equal for every test)	Control/Counter	UNSIGNED16	RW/R	No	Control the test behavior	
					Bit0	enable/disable the test 0: Test disabled 1: Test enabled
					Bit1 -7	Reserved for future use
					Bit8-15	Counter (indicates the number of test executions)

The tests can be enabled by writing the value 1 to Bit0 of the corresponding test object via SDO access another possibility is to make use of the test control object 0x2FFF (Table 10 and Table 11).

Table 10: Test Control Object

Attribute	Value
Index	0x2FFF
Name	Test control object
Object Code	ARRAY
Max SubIndex	16 (7if build for EL9800)

Table 11: Test Control Object Entries

Sub-Index	Description	Data Type	Access	PDO Mapping	Description / Default value	
1 – Max SubIndex (see Table 10)	Linked test object entry	UNSIGNED32	RW	No	Linked test object entry	
					Bit0 – 7	Reserved for future use
					Bit8 -15	Subindex of the test object entry
					Bit16-32	Index of the test object

The structure is similar to the PDO mapping objects. Every Subindex (entry) of the test control object maps an enable/disable function (Bit 0) to a physical digital input of the slave device.

If the application is compiled for the EL9800_4 EtherCAT Evaluation Kit the control object contains 7 entries (S11-7) which are mapped to the switches 2 to 8 (switch 1 is a global test function enable switch). If the stack is not compiled for the EL9800 the control element contains 16 Entries which are mapped to the GPO register (0xF10:0xF11).

A release build of the test application for the EL9800 EtherCAT Evaluation board is located in “SSC_Vxixx/hex”. To create new test application slave files select the configuration “TestApplication (EL9800_4A)” in the SSC Tool.

9.1 ESM Tests (0x2000 – 0x200F)

Table 12: Test Object 0x2000 (ESM Group 1)

Sub-Index	Description	Purpose
1	Invalid state transition from INIT to PreOP AL status Code 0x16 (Invalid Mailbox)	
2	Invalid state transition from PreOP to SafeOP AL status Code 0x1D (Invalid Output SyncManager config)	
3	Invalid state transition from PreOP to SafeOP AL status Code 0x1E (Invalid Input SyncManager config)	

9.2 CoE Tests (0x2020 – 0x202F)

Table 13: Test Object 0x2020 (CoE Group 1)

Sub-Index	Description	Purpose
1	Set Segmented indication on SDO upload if object is greater than 40Byte	
2	Create diagnosis message on every state change.	
3	Create diagnosis message with every application cycle.	

9.3 Generic Objects

The objects described in this chapter are used to test the correct SDO handling by the slave and the master. Table 14: Generic Objects includes all defined objects.

Table 14: Generic Objects

Index / Subindex	Deftype/Code	Access	Description	Purpose
0x3000	OCTED_STRING / VAR	RW	The object size is 349 Byte. If a 128 Bytes mailbox is used the object is transmitted with one complete and two segmented services minus one Byte. Default value: Each Byte is incremented by one (starts with 0x00)	This object is used to test the mailbox unlock mechanism within the slave stack and the correct segmented handling by the master.
0x3001	REAL32 / VAR	RW	Default value: 0xBABABABA	Use to test the value view by the master.
0x3002	UINT32 / VAR	RW	Default value: 0xBABABABA	
0x3003	RECORD / REC		Includes base datatypes less or equal 1Byte.	
SI1	BOOLEAN	RW	Default value: TRUE	
SI2	BIT1	RW	Default value:0x0	
SI3	BIT2	RW	Default value:0x3	
SI4	BIT3	RW	Default value:0x5	
SI5	BIT4	RW	Default value:0xA	
SI6	BIT5	RW	Default value:0x1A	
SI7	BIT6	RW	Default value:0x2A	
SI8	BIT7	RW	Default value:0x6A	
SI9	3BIT Align		Bit entries shall not overwrite WORD borders	
SI10	BIT8	RW	Default value:0xFF	
SI11	8BIT Align			

SI12	UINT8	RW	Default value:0xAA
SI13	INT8	RW	Default value:0xBB
0x3004	RECORD / REC		Object to test alignment, empty entries and several access rights
SI1	UINT16	RW	Default value:0x0
SI2	UINT16	RO	Default value:0x0
SI3	UINT8	RW	Default value:0x0
SI4	UINT8	RO	Default value:0x0
SI5	Empty		
SI6	BOOLEAN	RW	Default value:0x0
SI7	BOOLEAN	RO	Default value:0x0
SI8	14Bit Align		Default value:0x0
SI9	UINT8	WO	Default value:0x0
SI10	8Bit Align		Default value:0x0

10 Tool

The Slave Stack Code Tool allows creating new slave files depending on user specific requirements and settings.

List of slave files:

- C source code files
- Source code documentation (optional)
- Device Description (ESI) (optional)

Supported OS: Windows XP, Vista, 7 (32bit)

Required Framework: .NET (4.0)

Two new file extensions are registered: SSC Configuration File (*.escfg) and Slave Project File (*.esp). The configuration file is provided with each SSC version and includes all settings and information about the code. The Slave Project File is created by the configurator to save a slave project.

The main user interface (Figure 23: Configurator Main User Interface) is structured in the tool bar (*File, Project, Tool and Help*) above and 3 separated windows (*Slave Project Navigation, Slave Settings and Conflicts*).

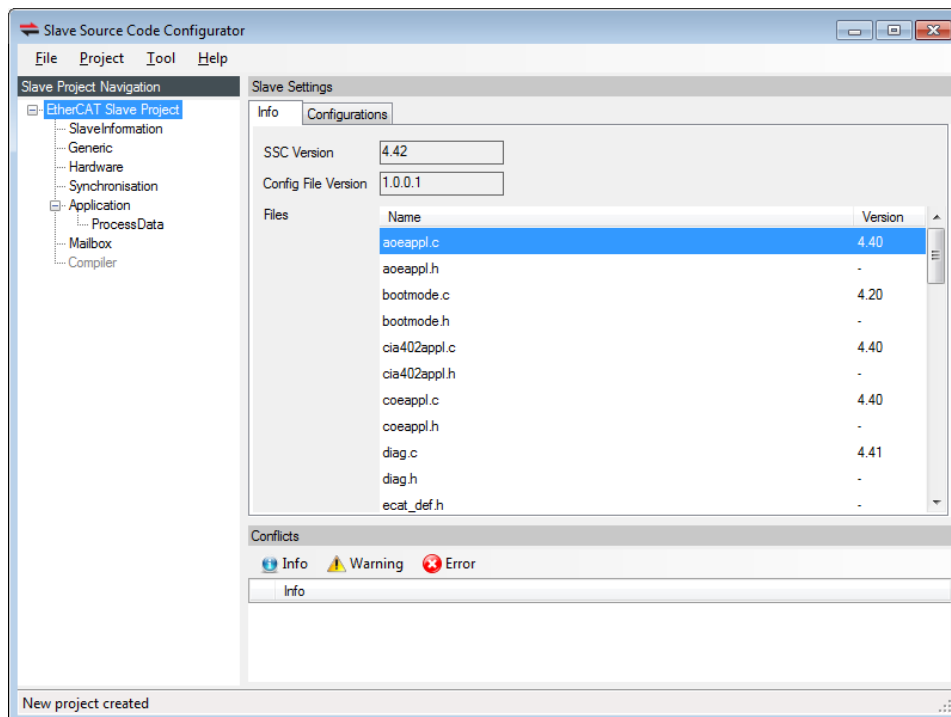


Figure 23: Configurator Main User Interface

The following chapters describe the elements in more detail.

10.1 Default Startup Dialogs

[Usage Information]

SSC tool usage information which need to be acknowledged before start working.

[Vendor Information]

Dialog to enter your Vendor Information including Vendor ID and Vendor name. These Information will be added automatically to a new slave project. This information can also be added via the Options dialog.

[Run Wizard]

Start the project wizard to create a new slave project.

10.2 Main User Interface Elements

10.2.1 Tool Bar

10.2.1.1 File

The file menu (Figure 24: Configurator File Menu) contains the project file operations.

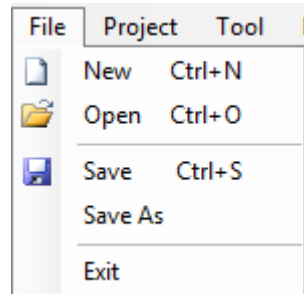


Figure 24: Configurator File Menu

[New]

Create a new slave project based on the local SSC files. The files are located in the Application Data folder. See 10.5Local SSC Update how to update the local SSC files.

[Open]

Open an existing project file.

[Save]

Save the actual settings to the current project file (if no project file was created before a file browser dialog appears to create a new file).

[Save As]

Save the actual settings to a new project file.

[Exit]

Close current session.

10.2.1.2 Project

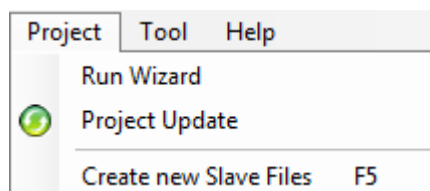


Figure 25: Configurator Source Menu

[Run Wizard]

Start the project wizard (see 10.3).

[Project Update]

Check for new Slave Stack Code version and update the current project. For further information see chapter 10.6.

[Create new Slave Files]

Open a dialog to create new slave files depending on the actual settings.

10.2.1.3 Tool

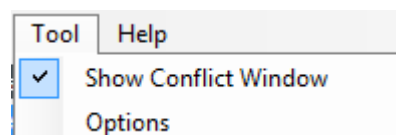


Figure 26: Configurator Tool Menu

[*Show Conflict Window*]
Hide or show the conflict window.

[*Options*]
Open the options dialog (see Figure 27: Configurator Options).

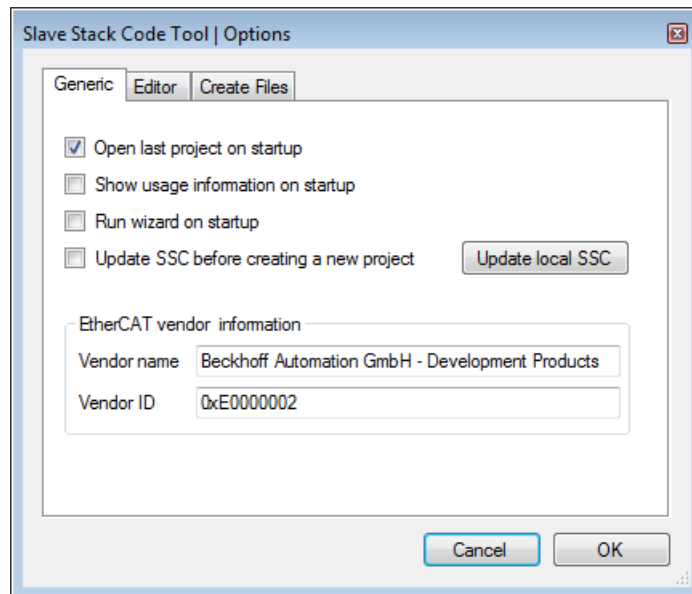


Figure 27: Configurator Options Window

- **Generic**

- [*Open last project on startup*]
The last slave project is reloaded on the next startup of the tool.
- [*Show usage information on startup*]
The tool usage information is shown on every startup (the usage information can also be shown by [*Help*] -> [*Usage Information*]).
- [*Run wizard on startup*]
The project wizard is started on the next tool startup. The project wizard can also be started by [*Project*] -> [*Run Wizard*].
- [*Update SSC before creating a new project*]
If this flag is set the Slave Stack Code Tool checks if a new SSC version is available before creating a new project.
NOTE: Before each project update ([*Project*] -> [*Project Update*]) an update of the local SSC file will be triggered.
The local SSC files are stored in the application data folder.
- [*Update local SSC*]
Update local SSC files.
- [*Vendor Name*]
Add your Vendor name here. This information will be added to slave project.
- [*Vendor ID*]
Add your Vendor ID here. This information will be added to the slave project.
If you don't have a Vendor ID yet please contact info@ethercat.org.

- **Editor**

- [*Show advanced settings*]
Show also settings which are marked as advanced (e.g. compiler settings)
- [*Show read only settings*]
Settings which are marked as read-only are shown (but remain read only). Read only settings are marked with a lock symbol (see Figure 28: Configurator Locked Setting).

ECAT_TIMER_INT	0
<input checked="" type="checkbox"/> INTERRUPTS_SUPPORTED	1

Figure 28: Configurator Locked Setting

- Create Files
 - o *[Add comment if obsolete code was skipped]*
Specify if a source code comment shall be added when code was deleted in comparison with the default Slave Stack Code.
 - o *[Create documentation]*
Create a code documentation based on the previously created source files. This feature requires external tools which are NOT covered by the SSC Beckhoff license agreement! Note the tool specific usage license.
Basically Doxygen is required. It is possible to use an specific Doxygen configuration file otherwise the default configuration is used which additionally requires GraphViz and HTML Help Workshop.
 - o *[Use configuration file]*
Use a user specific doxygen configuration file.
 - o *[Required Software]*
Select location of the required software.
 - o *[Create device description (ESI)]*
If this option is checked an application specific device description will be created.

10.2.1.4 Help

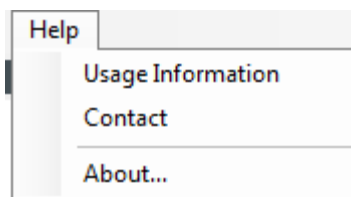


Figure 29: Configurator Help Menu

- [Usage Information]*
Show usage information and legal notes referring to the usage of the SSC Tool.
- [Contact]*
Create a new email to EtherCATSSC@beckhoff.com with your standard email client.
- [About]*
Show information about the SSC Tool.

10.2.2 Windows

- I. *[Slave Project Navigation]*
The *Slave Project Navigation* window lists all kinds of settings which can be configured. Selecting one of the nodes, the corresponding settings are displayed in the *Slave Settings* window. If the project was saved at least once the root node name is equal to the project name. This window also supports Drag & Drop.

[Slave Settings]
The information shown in this window depends on the selected node in the *Slave Project Navigation* window.

If the root project node is selected two tabs are available:

The *Info* tab includes the *SSC Version*, *Config File Version*, a list of all Slave Stack Code file which are evaluated by the configurator and user files (see Figure 30: Configurator Project Information). The file list within provides a right-click context menu (see Figure 31: Configurator File Context Menu).

The *Configuration* tab includes a list of several preconfigured settings, e.g for the EL9800 Evaluation board or the FC1100 application.

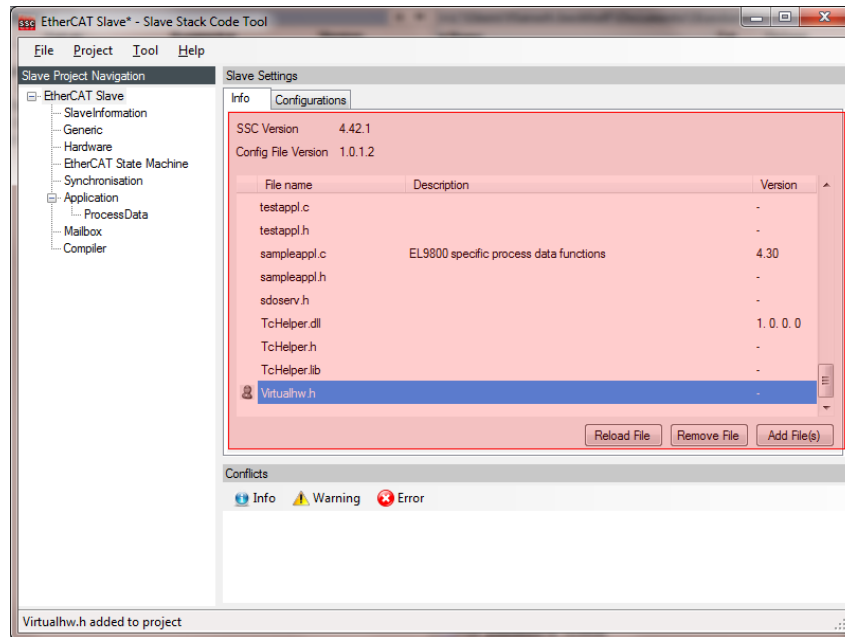


Figure 30: Configurator Project Information

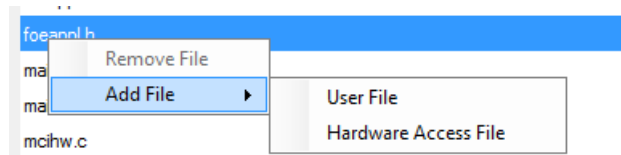


Figure 31: Configurator File Context Menu

[Reload File]

Cached file will be reloaded.

[Remove File]

Remove file from project.

[Add File(s)]

Add one or more files to the project. These files will not be evaluated and just copied to the output folder. If a new hardware access file is added it can be included to the generic files by defining "HW_ACCESS_FILE" in the hardware settings.

If one of the setting nodes are selected within the *Slave Project Navigation*, the corresponding settings are displayed within the *Slave Settings* window (Figure 32: Configurator Slave Settings).

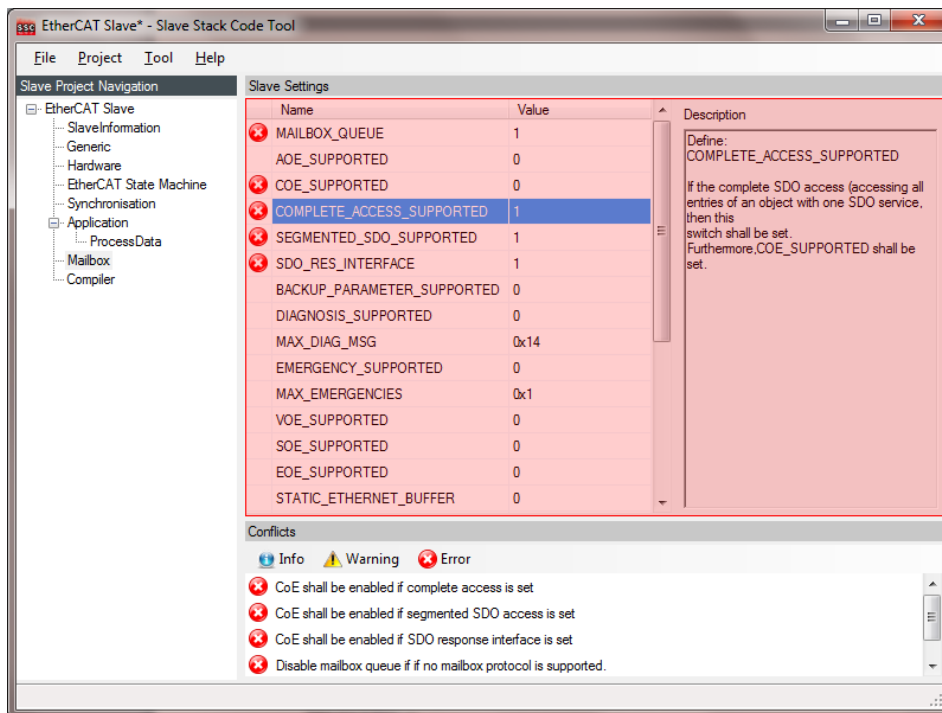


Figure 32: Configurator Slave Settings

II. Conflict Window

This window displays conflicts of settings. Conflicts are distinguished between errors, warnings and information. Conflicting settings are denoted by the corresponding conflict symbol.

NOTE: Not every combination of settings is checked. So it is up to the user to create a logical configuration.

[Error] Indicate that the configurator cannot create a valid slave stack with the actual configuration.

[Warning] The settings marked with the warning symbol should be checked before creating new slave files.

[Info] Additional information about the current configuration.

10.3 Project Wizard

The project wizard provides a step by step slave code configuration. The wizard is started either by [Source] Menu -> [Run Wizard] or on configurator startup (see option description in 10.2.1.3 Tool).

I. Select Configuration

In this step a project file and a predefined configuration need to be selected (Figure 33: Configurator Wizard Predefined Configurations). Predefined configurations are a list of settings.

- a. Specify location and name of new project file.
- b. Select a predefined configuration via drop down menu. A description about the selected configuration is shown below.
- c. Apply the selected configuration by clicking on continue button. The wizard will proceed with Step III.

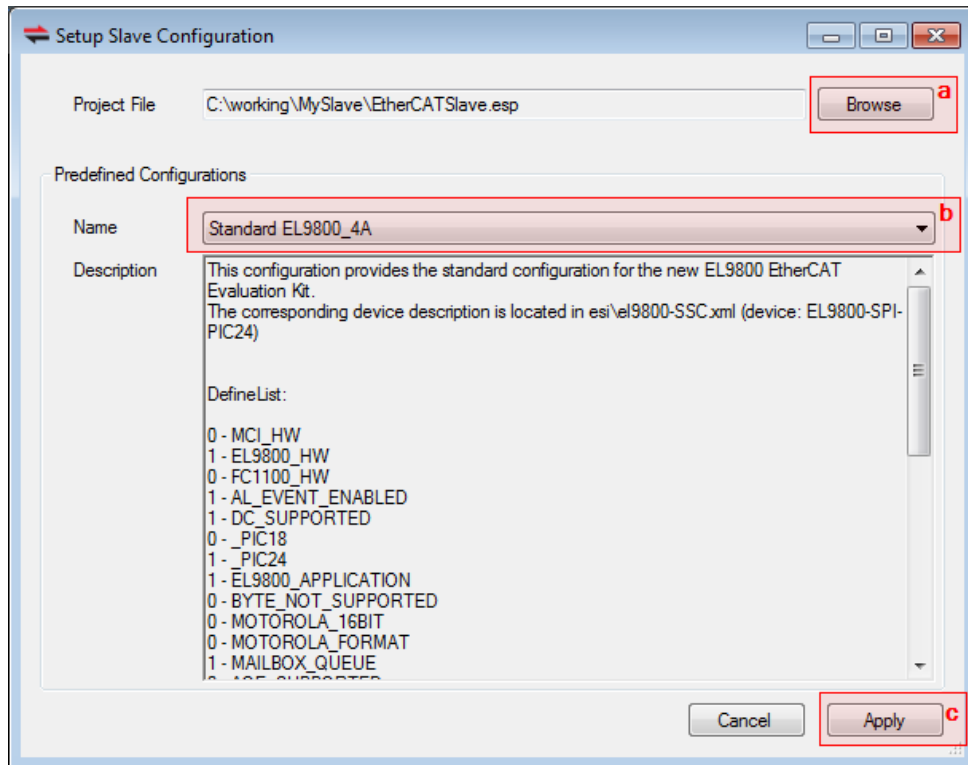


Figure 33: Configurator Wizard Predefined Configurations

[Browse]

Specify a project file.

[Cancel]

Close this dialog without saving the project and applying the selected configuration.

[Apply]

Apply selected configuration save the project and proceed with the next wizard step.

II. Create Slave Files

The Create Slave Files dialog is described in chapter 10.4.

10.4 Create Files

Creating new files is the last step to create a new slave stack. The dialog is opened by selecting [Project] -> [Create new Slave Files].

- a) Select output folder.
- b) [Start] Start creating new files.
- c) Output window dumps progress information.

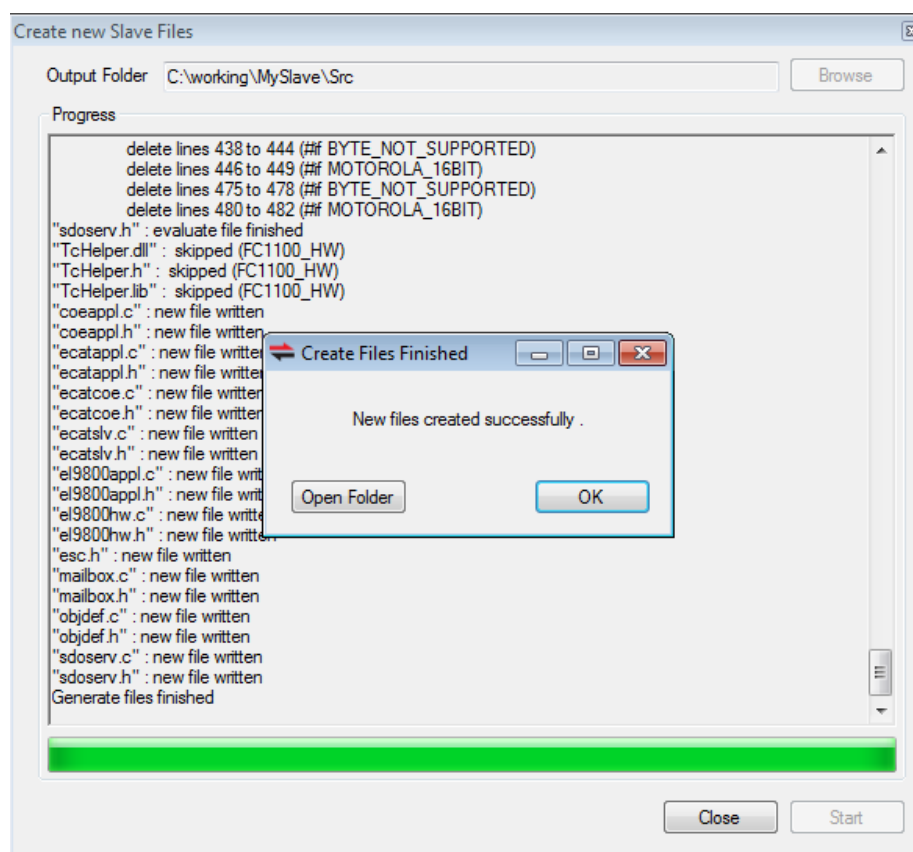


Figure 34: Configurator Create Files

[Browse]

Choose an output folder for the new slave files.

[Cancel]

Close dialog without creating new files

[Close]

Close dialog when new files were created.

[Start]

Create new slave files in the specified folder.

After new slave files are created a dialog appears to open the specified output folder or to return to the configurator.

10.5 Local SSC Update

Each new project is based on the local Slave Stack Code files. These files will be updated before a project update is started or when a new project is created (if this option is set).

The Slave Stack Code files are stored in application folder of the SSC Tool.

The update dialog is shown in Figure 35: Configurator Slave Stack Code Update.

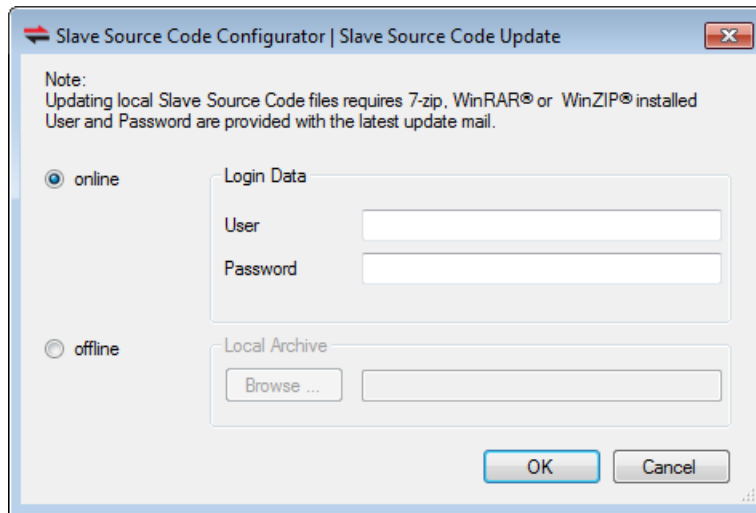


Figure 35: Configurator Slave Stack Code Update

[online]

Load SSC files from the Beckhoff FTP server. For login data see the latest “SSC update mail” or contact EtherCATSSC@beckhoff.com.

[offline]

Load SSC files from a local SSC zip archive.

Both options require an external archive tool (7-Zip, WinRAR® or WinZip®).

10.6 Project Update

The SSC Tool provides the possibility to update generic files within the current project to the latest version.

If new files are available the update dialog is show (Figure 36: Configurator Project Update Dialog).

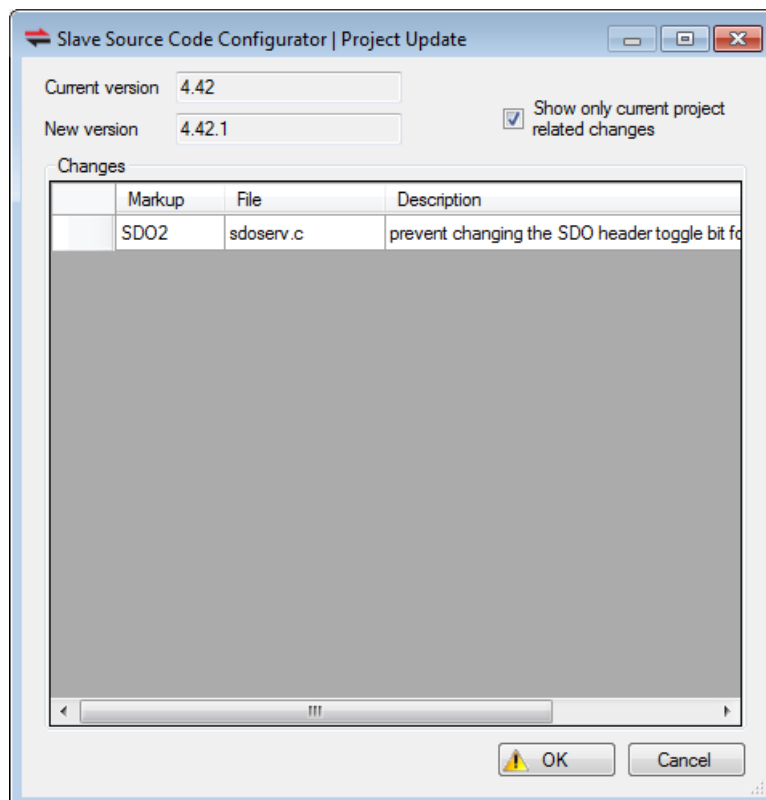


Figure 36: Configurator Project Update Dialog

[*Current version*]

Slave Stack Code version of the current project

[*New version*]

Latest Slave Stack Code version available

[*Show only current project related changes*]

If checked only changes are shown which are related to the current project settings. Otherwise all changes are displayed.

NOTE: If checked only the latest changes are shown. Related changes which are older than one version are not displayed.

[OK]

Update current project. If an exclamation mark is shown important information needs to be acknowledged before the project is updated.

[Cancel]

Cancel project update

Appendix

Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: <http://www.beckhoff.com>

You will also find further documentation for Beckhoff components there.

Beckhoff Headquarters

Beckhoff Automation GmbH

Eiserstr. 5

33415 Verl

Germany

phone: + 49 (0) 5246/963-0

fax: + 49 (0) 5246/963-198

e-mail: info@beckhoff.com

web: www.beckhoff.com

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

world-wide support

design, programming and commissioning of complex automation systems

and extensive training program for Beckhoff system components

hotline: + 49 (0) 5246/963-157

fax: + 49 (0) 5246/963-9157

e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

on-site service

repair service

spare parts service

hotline service

hotline: + 49 (0) 5246/963-460

fax: + 49 (0) 5246/963-479

e-mail: service@beckhoff.com

EtherCAT Technology Group (ETG) Headquarters

Phone: +49 (911) 540 5620

Fax: +49 (911) 540 5629

Email: info@ethercat.org

Internet: www.ethercat.org