EtherCAT Slave Stack

Software Manual

to Product P.4520.01

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

N O T E

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

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

esd electronics gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

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

Document file:	I:\Texte\Doku\MANUALS\PROGRAM\EtherCAT\Slave\Englisch\EtherCAT_Slave_Manual_en_18.odt
Date of print:	2018-05-03

Software version:	EtherCAT Slave Stack Version >= 1.3.8

Document History

The changes in the document listed below affect changes in the software as well as changes in the description of the facts, only.

Revision	Chapter	Changes versus previous version	Date
1.0		First Release	2012-05-09
1.1	1.2.2	Added “EtherCAT Slave Controller”	2012-11-06
	-	Editorial changes (Note/Logo)	
1.2	-	Minor changes/clarifications for several sections	2013-05-07
	-	Added section “Reference“	
	2.6.61	Updated *ESS_CONFIGURATION*	
	2.4.25	Added *essODPDOParamCreate()* and other *essPDOParam* functions	
	2.4	Added *essGetTag()*	
	2.6.10	Added some *ESS_OD_ENTRY_FLAGS*	
	2.6.20	Member added to the *ESS_CBDATA_COE_EVENT* struct	
	2.5	Added note to *cbInputsUpdated*	
	4.1	Fig. 3 updated (Includes overview)	
	4.2	Added descriptions for several “*ESS_CFG_*” defines	
	2.5, 2.6	Added new callbacks *ESS_CB_EEPROM_EMULATION* and *ESS_CB_DC* and their data types	
	2.6	Added *ESS_STATISTICS*	
1.3	2.6.58	Added EoE frames to *ESS_STATISTICS*	2014-03-04
	4.2	Updated *essConfig.h* defines	
	2.2.9	Chapter VoE updated and renamed to AoE/SoE/VoE	
1.4	2.4.25	Added *essODDelete()*	2014-08-19
	2.4.5	Added remark to *essOpen()*/*essClose()* serialization	
1.5	-	Editorial changes	2014-01-05
	4	Added Stack/Application flow chart	
	2.4	*essSyncInputs()* and *essIoctl()* added, *essODSyncInputMappedEntries()* removed	
	2.6	*ESS_RESULT* and *ESS_OD_ENTRY_FLAGS* updated	
	4.3	*essHALInit()*/*essHALFinish()* now *essHALOpen()*/*essHALClose()*. *Added* *essHALIoctl()*.	
1.6	-	Editorial changes	
	1.2.2	Updated list of supported esd EtherCAT hardware	2016-07-04

Revision	Chapter	Changes versus previous version	Date
	4.1	Added description of supported *ESS_PLATFORM_XXX*	2016-07-12
	4.2	Added description of *CFG_HAL_USE_SYNCX_IRQ*	2016-07-04
	5	Updated list of supported esd EtherCAT hardware	2016-07-12
1.7	-	Editorial changes	
	2.3.2	Added new chapter about EtherCAT device identification.	2016-07-27
	2.5	Extended *cbStateRequest* to support device ID requests.	2016-07-27
	2.6.23	Extended *ESS_CBDATA_STATE_REQUEST* to support the 'Explicit Device ID' mechanism defined in ETG.1020.	2016-07-27
	4.2	Added *essConfig.h* define *CFG_ESS_EXPLICIT_DEVICE_ID*	2016-07-27
1.8	-	Editorial changes	
	2.3.1	Revised description for bootstrap mailbox support.	2017-04-07
	2.3.3	New chapter describing the PDO mapping.	2018-04-30
	2.3.4	New chapter describing the expected behavior of the local input/output data handler in SAFEOP and OP.	2018-01-10
	2.6.22	Description of mapping macros ESS_MAP_XXX.	2018-04-30
	2.6.24	Added description of ESS_SM_EVENT_FLAG_SAFE_OUTPUTS.	2018-01-10
	2.6.28	Added description of *maxDataLen* for ESS_CBDATA_FOE_OPEN	2017-05-17
	2.6.30	Added description of *maxDataLen* for ESS_CBDATA_FOE_DATA and how to indicate a FoE Busy situation	2017-05-17
	2.6.66	Documented additional AL Status Codes defined in ETG.1020.	2016-08-16
	2.6.70	Documented additional CoE Abort Codes defined in ETG.1020.	2017-02-15
	2.6.71	Documented additional FoE error codes defined in ETG.1020.	2017-03-09
	4.2	Added *essConfig.h* define *CFG_ESS_HAVE_XXX_FUNC*	2017-05-08

Technical details are subject to change without further notice.

Table of contents

Abbreviations and terms

Abbr.	Term	Description/Comment
ABI	Application Binary Interface	
ACK	Acknowledge	
AoE	ADS over EtherCAT	(ADS: Automation Device Specifications)
API	Application programming interface	
	Byte	8 bit
CoE	CAN application protocol over EtherCAT	
CTT	Conformance Test Tool	EtherCAT slave device conformance test. (Beckhoff Product: ET9400)
DIV	Device Identification Value	
DC	Distributed Clock	
DNS	Domain Name System	
	DWord	32 bit
EEPROM	Electrically Erasable Programmable Read-Only Memory	
EoE	Ethernet over EtherCAT	
ESC	EtherCAT Slave Controller	
ESI	EtherCAT Slave Information	Contains configuration infos, etc. about an EtherCAT slave device. As .xml file or within the slave's EEPROM
ET1100		Common ESC by Beckhoff Automation GmbH, see [ET1100]
ETG	EtherCAT Technology Group	Homepage: www.ethercat.org
Flash	Flash memory	
FMMU	Field bus Memory Management Unit	
FoE	File transfer protocol over EtherCAT	
GCC	GNU Compiler Collection	
GPL	GNU General Public License	
HAL	Hardware Abstraction Layer	
Init	EtherCAT device state "Init"	
IP	Internet Protocol	Referring only to IPv4
ISR	Interrupt Service Routine	
MAC	Media Access Control	
MBox	Mailbox	EtherCAT slave device Mailbox
OD	Object Dictionary	CoE object dictionary
Op	EtherCAT device state "Operational"	
PCIe	PCI Express (Peripheral Component Interconnect Express)	
PDI	Process Data Interface	
PDO	Process Data Object	
PreOp	EtherCAT device state "Pre-Operational"	
Rx	Receiver	Direction: From master/EtherCAT to slave
SafeOp	EtherCAT device state "Safe-Operational"	
	Slave	EtherCAT slave device
SM	SyncMan, Synchronization Manager	
SoE	Servo Profile over EtherCAT	
SPI	Serial Peripheral Interface	
Tx	Transmitter	Direction: From slave to master/EtherCAT
VoE	Vendor specific protocol over EtherCAT	
	Word	16 bit

Reference

[ET1100]	Beckhoff Automation GmbH, ET1100 Hardware Data Sheet, Version 2.9
[ESC.1]	Beckhoff Automation GmbH, EtherCAT Slave Controller, Section I - Technology, Version 2.2
[ESC.2]	Beckhoff Automation GmbH, EtherCAT Slave Controller, Section II – Register Description, Version 2.7
[ETG.1000.5]	EtherCAT Technology Group, EtherCAT Specification – Part 5, Version 1.0.3
[ETG.1000.6]	EtherCAT Technology Group, EtherCAT Specification – Part 6, Version 1.0.3
[ETG.1004]	EtherCAT Technology Group, EtherCAT Unit Specification, Version 1.0.0
[ETG.1020]	EtherCAT Technology Group, EtherCAT Protocol Enhancements, Version 1.2.0
[ETG.2010]	EtherCAT Technology Group, EtherCAT Slave Information Interface, Version 1.0.0
[ETG.2200]	EtherCAT Technology Group, EtherCAT Slave Implementation Guide, Version 2.1.7

1. Introduction

The esd EtherCAT Slave Stack offers an easy to use API to build complex EtherCAT Slave devices.

The stack is distributed as a reconfigured library which is referred to as Binary *Version* in this document or as source code which is referred to as *Source Version* (see chapter 1.2.2).

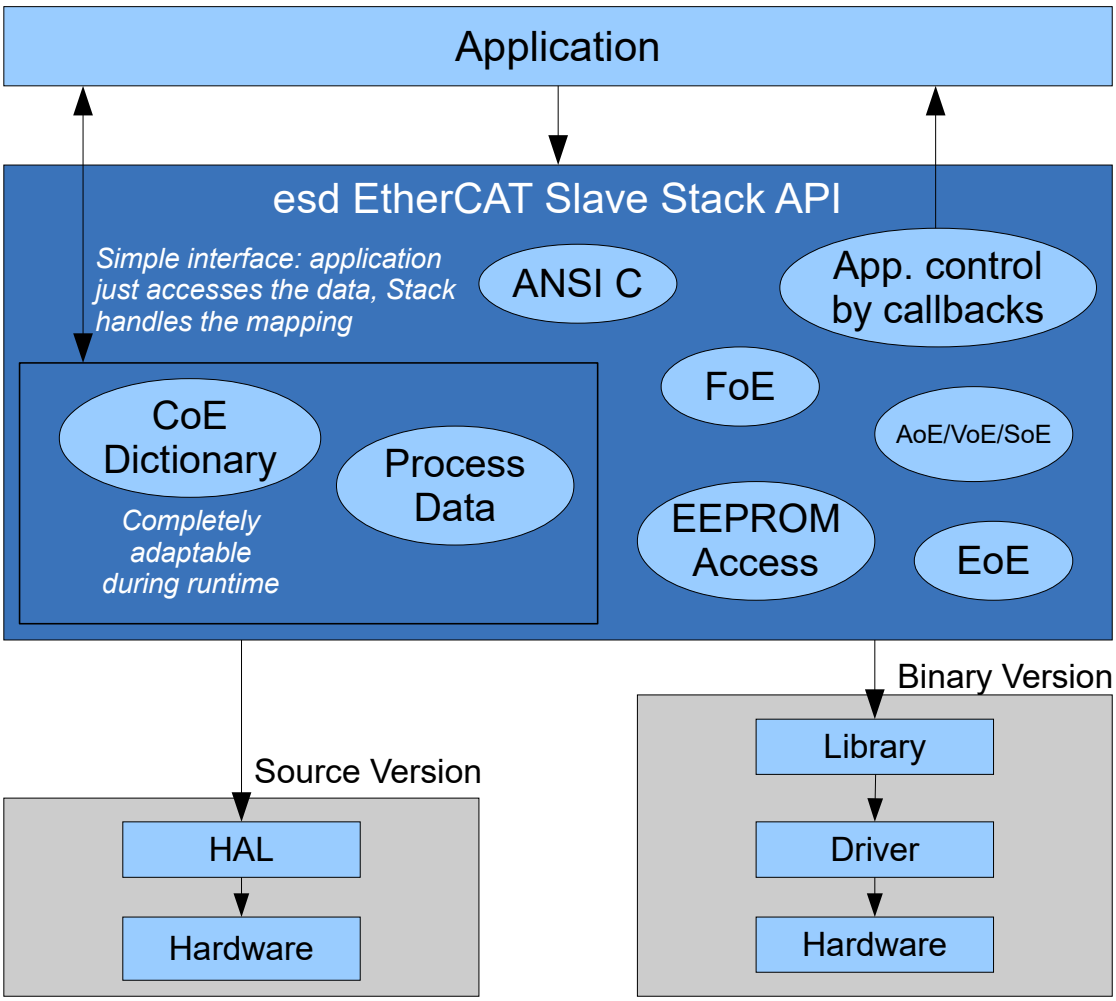

Fig. 1: Architecture overview

For implementation details / application flow chart, refer to the Source Code Version specific description in chapter 4.

1.1 Features

- Based on a stable API which hides complexity and hardware dependency behind common function calls and callbacks.
- Includes support for all major mailbox protocols
  - CoE includes “SDO Information Service”, “Segmented SDO Service”, dynamic PDO assignment and dynamic PDO configuration
  - FoE by callback for each data segment with busy indication support and support for extensions defined in [ETG.1020].
  - EoE with callback for completely assembled frames (from EtherCAT) and a simple function to send an Ethernet frame to EtherCAT (Stack handles fragmentation, etc.)
  - AoE/VoE/SoE by simple callback for each mailbox packet of that type
- Comprehensive support for CoE object dictionary and process data – application just accesses the objects and the stack handles almost everything, e.g. updates when they are PDO mapped, automatically
  - Dynamic dictionary, completely changeable during runtime.
  - Dynamic PDO mapping, completely changeable during runtime.
  - Support to map objects with more than 31 bytes according to [ETG.1020].
  - Entries can also be created without data pointer to provide data dynamically during the SDO access
  - Implicit handling of important entries, such as PDOs and PDO assignment objects.
  - Callbacks for important events, e.g. before and after SDO Download, etc.
- Support for the standard *Explicit Device ID* mechanisms according to [ETG.1020].
- Includes functions to read/write ESC’s E^2PROM
- Header for library in ANSI C
- Source code:
  - ANSI C
  - Well defined HAL to adapt to own hardware with as little effort as possible
  - Little/big endian compatible
- Always validated with the latest version of the EtherCAT Conformance Test Tool (CTT).
- Comprehensive example code application for a quick start.

1.2 Requirements

1.2.1 EtherCAT Slave development in general

Although this stack greatly simplifies the EtherCAT slave development you still have to be familiar with the ETG specifications to develop a slave application compliant to the EtherCAT standards. The ETG especially requires you to:

- test your slave against the CTT[1]
- be an ETG member with a valid member ID

For more information/requirements check the ETG homepage at www.ethercat.org (a good start is [ETG.2200])

1.2.2 esd EtherCAT Slave Stack

The esd EtherCAT Slave Stack (ESS) is available as a source code version which can be adapted to your custom EtherCAT hardware and operating system and as a binary version ready to be used with the various esd EtherCAT slave interfaces. The latter is configured for the target hardware with all supported EtherCAT protocols enabled.

Binary version

- These versions are usually bundled with hardware. The stack is delivered as a (shared) library and the HAL is already included.

  Available hardware:

  - **ECS-PCIe/1100:** (Order No. E.1100.02)
    - PCI Express card with ET1100 (only INTx support)
    - Includes Windows driver for XP/Vista/7/8/10 (32/64 bit)
    - Includes Linux driver (“UIO”) as source code under GPLv2
  - **ECS-PMC/FPGA:** (Order No. E.1104.02)
    - PMC card with EtherCAT IP Core (INTx / MSI support)
    - Includes Windows driver for XP/Vista/7/8/10 (32/64 bit)
    - Includes Linux driver (“UIO”) as source code under GPLv2
    - Includes VxWorks 7 (x86) VxBus Gen2 driver as loadable kernel module
  - **ECS-XMC/FPGA:** (Order No. E.1102.02)
    - XMC card with EtherCAT IP Core (INTx / MSI support)
    - Includes Windows driver for XP/Vista/7/8/10 (32/64 bit)
    - Includes Linux driver (“UIO”) as source code under GPLv2
    - Includes VxWorks 7 (x86) VxBus Gen2 driver as loadable kernel module

1 As this tool is enhanced and extended regularly, changes to the esd EtherCAT Slave Stack might also become necessary.

- **ECS-PCIe/FPGA / ECS-PCIe/FPGA-LP:** (Order No. E.1106.02 / E.1106.04)
  - PCI Express card with EtherCAT IP Core (INTx / MSI support)
  - Includes Windows driver for XP/Vista/7/8/10 (32/64 bit)
  - Includes Linux driver (“UIO”) as source code under GPLv2
  - Includes VxWorks 7 (x86) VxBus Gen2 driver as loadable kernel module

Source Code Version

- Requires customization of the HAL to the target system, see section 4
- Sample resource usage for an EtherCAT slave application[2]: (Target: ARM Cortex™-M3, 20 kB RAM total, FreeRTOS™)
  - RAM: approx. **5 kB** (BSS/DATA/Heap) + **2 kB** (Stack)
  - ROM: approx. **15 kB** (CODE/CONST)
- EtherCAT Slave Controller
  - Designed for ET1100 and compatible
  - Texas Instruments Programmable Real Time Units (PRU) also supported. (AM335x, Code Composer Studio project file exists)

2 64 byte process data, CoE with “SDO Information Service” support and approx. 30 objects total.

2. API

2.1 Usage overview

- To access the API only `ess.h` has to be included
- To start the stack the functions *`essOpen()`* and *`essStart()`* have to be called
  - *`essOpen()`* selects and initializes the underlying hardware
  - *`essStart()`* then gives the application control to the stack, i.e. this function usually never returns
- Application is now driven by the stack's callback handler. This can be a cyclic callback or certain other callbacks triggered by EtherCAT interrupts (see 2.5)
- All functions (except those to start/stop the stack, of course) may be called only when the stack is running, i.e. during its callbacks (for certain functions more/less restrictions might exist, see section 2.4)
- It's transparent to the application whether the underlying HAL is connected to a real interrupt handler or just polls the ESC registers
  - Therefore application code is extremely hardware/configuration independent
  - But if the application consists of multiple threads/tasks you must not call any esd EtherCAT Slave Stack function from a thread/task other than the one that called *`essOpen()`* (Unless different devices are handled, see also *`essOpen() devIdx`* Parameter)
- The *`config`* parameter of *`essOpen()`* allows to configure some of the mentioned behavior, see 2.4.4

(For details see also Source Code Version specific information in chapter 4)

2.2 Quick Start

This section shall only give a more detailed overview of the API usage as a quick start. F, for a detailed function description see section 2.4. Complete example applications are provided with the stack's *.../apps/* directory[3]:

- *complex.c* handles multiple variables with the CoE dictionary and shows different flags/callbacks, etc. Also includes FoE samples.

  This should be the base/template for your application
- *eoe.c* shows examples to handle EoE

2.2.1 Opening a device

This is done with *essOpen()*:

```
ESS_RESULT res;
res = essOpen(0, &config, &hDev);
if (res != ESS_RESULT_SUCCESS) {
    PRINT(("Opening device failed with %s",
           essFormatResult(res, ESS_FORMAT_SHORT)));
    exit(10);
}
```

Used variables

```
static ESS_HANDLE hDev;
```

This is the handle to the device that was opened, almost all stack functions need this to distinguish multiple devices. See also *ESS_HANDLE*.

```
static const ESS_CONFIGURATION config = { ...
```

This contains the configuration for the slave that is developed. It contains stack configuration such as options for the HAL, slave configuration like the SM configuration and the application callbacks. See *ESS_CONFIGURATION* for complete information.

Note that the configuration (especially the SM configuration) given to the stack must match the slave configuration that is stored in the ESI (.xml and EEPROM).

3 "*...*" refers to your stack installation directory.

2.2.2 Starting the stack

This is done with *essStart()*: (only after a successful call to *essOpen()*, of course)

```
res = essStart(hDev);
if (res != ESS_RESULT_SUCCESS)
    PRINT(("essStart() returned %s", essFormatResult(res, ESS_FORMAT_SHORT)));
```

Unless there is no error that prevents the stack from starting, *essStart()* does not return – i.e. the stack takes control over the application.

To execute application code, the stack's callbacks must be used.

2.2.3 Running application code

In *essOpen()* we also set the callbacks: functions that are called by the stack whenever something happens. Only there application code can be executed[4].

One of it is the cyclic callback, this function is just called cyclically, it could look like this:

```
static void cbCyclic(ESS_CBDATA_CYCLIC* cbData)
{
    HandleApplicationCode();

    if (ApplicationDetectedCriticalError())
        essStop(hDev);
}
```

As described in the previous section: *essStart()* usually does not return. So when the application wants to stop the stack it has to call *essStop()* from this cyclic callback.

A table with the available callbacks can be found in section 2.5.

2.2.4 Creating the CoE object dictionary

This is done with *essODCreate()*. (Before the stack is started, but after the device was opened)

```
ESS_RESULT res = essODCreate(hDev, ESS_OD_FLAGS_HANDLE_SM_TYPES);
if (res != ESS_RESULT_SUCCESS)
    exit(20);
```

By the *ESS_OD_FLAGS_HANDLE_SM_TYPES* flag the stack will automatically create the object 0x1c00 for us (which is mandatory, so we have to create it to be EtherCAT compliant).

4 If your application consists of multiple threads/tasks don't access any stack function from there.

2.2.5 Adding objects to the CoE dictionary

To manually add an object (which consists of one or more entries) *essODObjectAdd()* and *essODEntryAdd()* are used.

In the example below a dictionary entry at index 0x2000 and subindex 0 with the name To create an object 0x2000 named “Output1” as that is a 32 bit unsigned integer which can be read or written is created:

```
static const ESS_OD_OBJECT_INFOS objInfos =   { "Output1", COE_DATATYPE_UDINT,
                                                COE_CODE_VARIABLE };
static const ESS_OD_ENTRY_INFOS entryInfos0 = { "Output1", NULL, NULL, NULL,
                                                0, COE_DATATYPE_UDINT };

res = essODObjectAdd(hDev, 0x2000, ESS_OD_OBJECT_FLAGS_NONE, &objInfos);
if (res != ESS_RESULT_SUCCESS) goto error;

res = essODEntryAdd(hDev, 0x2000, 0x00, 32, &output1, COE_ACCESS_RW,
                    ESS_OD_ENTRY_FLAGS_NONE, &entryInfos0);
if (res != ESS_RESULT_SUCCESS) goto error;
```

All object and entry descriptions returned with the *SDO Info Service* are created implicitly. Please note that the EtherCAT stack does not support segmented replies for this service. To avoid truncated object or entry descriptions (and discrepancies to the ESI file) make sure that the minimum mailbox size is sufficient to return the longest string provided in the parameter *name* of ESS_OD_OBJECT_INFOS or ESS_OD_ENTRY_INFOS.

2.2.6 Accessing CoE entry data

As only a pointer to the entry’s data is given to the stack, some actions have to be taken to “synchronize” the access between the application and the stack.

1. **Input variables**

   When data for an entry that might be mapped into input process data is changed, the stack needs to know this, use *essSyncInputs()*.

2. **Large entries that require segmented transfer**

   As a segmented transfer takes multiple cycles, the application will be called within the transfer, i.e. the application might access data to an entry that was written only partially or is currently uploaded to the master.

   Use the *ESS_OD_ENTRY_CALLBACK* info of the *ESS_CB_COE_EVENT* callback to detect start and end of a download/upload: Make sure a long entry is not accessed after a

   ESS_OD_ENTRY_CALLBACK_STARTING_DOWNLOAD/UPLOAD

   callback was received before either

   ESS_OD_ENTRY_CALLBACK_COMPLETED_DOWNLOAD/UPLOAD or

   ESS_OD_ENTRY_CALLBACK_ABORTED_DOWNLOAD/UPLOAD is received.

2.2.7 FoE

FoE is used by setting these callback members in the *ESS_CONFIGURATION/ESS_CALLBACKS* structure: (Must be enabled in *essConfig.h* by *CFG_ESS_SUPPORT_FOE*)

1. *cbFoEOpen* (Type definition: *ESS_CB_FOE_OPEN*)
2. *cbFoEClose* (Type definition: *ESS_CB_FOE_CLOSE*)
3. *cbFoEData* (Type definition: *ESS_CB_FOE_DATA*)

See section 2.5 and the type definition's parameter description for details.

Examples are provided in *.../apps/complex.c*.

2.2.8 EoE

EoE is used by setting these callback members in the *ESS_CONFIGURATION/ESS_CALLBACKS* struct: (Must be enabled in *essConfig.h* by *CFG_ESS_SUPPORT_EOE*)

1. *cbEoESetIPParam* (Type definition: *ESS_CB_EOE_SETIPPARAM*)
2. *cbEoESetAddrFilter* (Type definition: *ESS_CB_EOE_SETADDRFILTER*)
3. *cbEoEFrame* (Type definition: *ESS_CB_EOE_FRAME*)

See section 2.5 and the type definition's parameter description for details.

To send an Ethernet frame to EtherCAT *essEoESendFrame()* is used. Examples are provided in *.../apps/eoe.c*.

2.2.9 AoE/SoE/VoE

These three protocols are handled the same way: set the callback members in the *ESS_CONFIGURATION/ESS_CALLBACKS* struct and enable *CFG_ESS_SUPPORT_*x*OE* in *essConfig.h*).

See *ESS_CB_AOE*, and *ESS_CBDATA_AOE* for details.

The AoE/VoE/SoE packet can be answered only in this callback: modify the mailbox packet at *cbData->mb* as needed an set *cbData->result* to *MBX_ERR_SUCCESS* to send this reply.

These protocols do not have extended support by the stack, i.e. the stack handles the mailbox protocol but not the AoE/SoE/VoE protocol itself. Examples in *.../apps/complex.c* show how to get started.

2.3 Configuration

2.3.1 Sync Manager

The application has to define the Sync Manager (SM) configuration as an array of `ESS_SM_CONFIGURATION` structures which is referenced by `ESS_CONFIGURATION`. The latter contains the number of entries for the *Standard Configuration* followed by two optional *Bootstrap Configuration* entries.

The table below shows the expected/typical assignment of the *Standard Configuration* for an EtherCAT slave device with and without mailbox support. Additional SM configurations for process data may follow. The number of available SMs depends on the ESC and has to be defined (at compile time) as `CFG_ESS_MAX_SM_COUNT`. The number of entries for the *Standard Configuration* in the `ESS_SM_CONFIGURATION` array has to be set in the variable *`smConfigCount`* of `ESS_CONFIGURATION`.

Sync Manger (SM)	Device with Mailbox	Device w/o Mailbox
SM0	Mailbox Out	Output Process Data
SM1	Mailbox In	Input Process Data
SM2	Output Process Data	
SM3	Input Process Data	

Table 1: Default SyncManager Assignment

Please refer to [ETG.1020] for further default assignments if eg. the device has no output process data.

The mandatory `ESS_SM_CONFIGURATION` array entries of this *Standard Configuration* may optionally follow two entries for a special *Bootstrap Mode* mailbox configuration (see 2.3.1.2). In this case the variable *`smConfigCountBootstrap`* of `ESS_CONFIGURATION` has to be set to 2 without a bootstrap configuration to 0.

Each SM configuration is defined by it's physical start address, a SM type specific control byte and the definition of the default, minimum and maximum size. **These parameter has to correspond to the values defined in the SII and ESI file**.

The EtherCAT stack performs several checks based on these values if the EtherCAT master assigns/changes the SM configuration:

- The assigned size is checked that it does not come below the configured minimum value.
- The assigned size is checked that id does not exceed the configured maximum value (if this is not set to 0).
- The assigned start address has to match the configured start address (if this is not set to 0).
- The assigned control byte (mailbox type) has to match the configured control byte.
- The stack checks that the sum of assigned values for start address and size of all SMs do not result in overlapping ESC DPRAM areas.If one of the checks does not succeed the slave device will usually not perform the requested state change.

Note: The default SM size is currently unused by the slave stack and the maximum value of any SM configuration must not exceed the value defined for `CFG_ESS_MAX_MBX_LEN`.

2.3.1.1 Standard Mailbox

The configuration of the mailbox sizes is always a trade-off between mailbox protocol throughput/performance of your slave and the overall use of Ethernet bandwidth as an EtrherCAT master always has to read the complete mailbox. A mailbox size configuration of 1024 bytes would require already about 10 microseconds additional cycle time. The latter is especially *expensive* if the mailbox is cyclically polled by the master because the *Write Event Flag* of the Input Mailbox is not mapped to the process image with the help of one FMMU.

Example:

```
#define SLAVE_MBX_OUT_DEF     522
#define SLAVE_MBX_OUT_MAX     1024
#define SLAVE_MBX_IN_DEF      522
#define SLAVE_MBX_IN_MAX      1024

static const ESS_SM_CONFIGURATION smConfigs[] =
{
    { /* SM0: MBoxOut */
        46, /* minSize */
        SLAVE_MBX_OUT_DEF,
        SLAVE_MBX_OUT_MAX,
        0x1000, /* startAddr */
        SM_TYPE_MBXOUT | REG_MASK_SMCONTROL_PDIINT /* contrByte */
    },

    { /* SM1: MBoxIn */
        46, /* minSize */
        SLAVE_MBX_IN_DEF,
        SLAVE_MBX_IN_MAX,
        0x1400, /* startAddr */
        SM_TYPE_MBXIN | REG_MASK_SMCONTROL_PDIINT /* contrByte */
    },

    { /* SM2: Outputs */
        0, /* minSize */
        4, /* defSize */
        0, /* maxSize (0: unchecked) */
        0x1800, /* startAddr */
        SM_TYPE_OUTPUTS | REG_MASK_SMCONTROL_PDIINT /* contrByte */
    },

    { /* SM3: Inputs */
        0, /* minSize */
        4, /* defSize */
        0, /* maxSize (0: unchecked) */
        0x2400, /* startAddr */
        SM_TYPE_INPUTS | REG_MASK_SMCONTROL_PDIINT /* contrByte */
    }
};
```

2.3.1.2 Bootstrap Mailbox

If your application supports the FoE protocol for firmware updates you can define a special (larger) mailbox which is only applied in the mode BOOTSTRAP.

Example:

```
#define SLAVE_MBX_OUT_DEF  522
#define SLAVE_MBX_OUT_MAX  1024
#define SLAVE_MBX_IN_DEF   522
#define SLAVE_MBX_IN_MAX   1024

static const ESS_SM_CONFIGURATION smConfigs[] = {
     { /* SM0: MBoxOut (Standard) */
         46, /* minSize */
         SLAVE_MBX_OUT_DEF,
         SLAVE_MBX_OUT_MAX,
         0x1000, /* startAddr */
         SM_TYPE_MBXOUT | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     },

     { /* SM1: MBoxIn (Standard) */
         46, /* minSize */
         SLAVE_MBX_IN_DEF,
         SLAVE_MBX_IN_MAX,
         0x1400, /* startAddr */
         SM_TYPE_MBXIN | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     },

     { /* SM2: Outputs */
         0, /* minSize */
         4, /* defSize */
         0, /* maxSize (0: unchecked) */
         0x1800, /* startAddr */
         SM_TYPE_OUTPUTS | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     },

     { /* SM3: Inputs */
         0, /* minSize */
         4, /* defSize */
         0, /* maxSize (0: unchecked) */
         0x2400, /* startAddr */
         SM_TYPE_INPUTS | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     }

     { /* SM0: MBoxOut (Bootstrap) */
         SLAVE_MBX_OUT_MAX, /* minSize */
         SLAVE_MBX_OUT_MAX,
         SLAVE_MBX_OUT_MAX,
         0x1000, /* startAddr */
         SM_TYPE_MBXOUT | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     },

     { /* SM1: MBoxIn (Bootstrap) */
        SLAVE_MBX_IN_MAX, /* minSize */
        SLAVE_MBX_IN_MAX,
        SLAVE_MBX_IN_MAX,
        0x1400, /* startAddr */
        SM_TYPE_MBXIN | REG_MASK_SMCONTROL_PDIINT /* contrByte */
     },
};
```

2.3.2 Device Identification

EtherCAT supports the explicit identification of a device which is used by the master for the detection of a Hot Connect stations or to prevent an unambiguous swap of devices. The unique Device Identification Value (**DIV**) in the range from 0..65355 has to be stored in non-volatile memory or has to be derived from a non-volatile selector (e.g. DIP switch). At the moment three different methods for the device identification are supported:

- **Second Slave Address (SSA) / Alias Address:** The DIV is stored in the E^2PROM of the EtherCAT slave and is loaded autonomously by the ESC into register 0x0012 where it can be used by the master for the identification. The latter is only performed after power-on reset [ESC.2]. For this reason [ETG.1020] defines a *Device Identification Reload Object* which allows reloading register 0x0012 from E^2PROM without a power cycle. The example provided in *.../apps/complex.c*. implements this object if `ESS_COMPLEX_ID_RELOAD_OBJECT` is defined. The configuration of the ID has to be performed with a configuration tool (e.g. the esd EtherCAT Workbench).
- **Data Word Identification Mode / Direct ID:** The DIV is stored in the process data area of the EtherCAT slave controller. [ETG.1020] restricts the possible memory ranges to 0x0F18..0x0F1F and 0x1000..0x1003. The address offset implemented by the slave has to be indicated to the master in the *Esi:Info:IdentificationAdo* element and/or the related SII elements [ETG.2010]. This method is often used by simple slaves without an MCU which connect an ID-Selector directly with the digital I/O inputs of an ESC but can also be enabled in *.../apps/complex.c*. by defining a valid data word address with `ESS_COMPLEX_DIRECT_ID_ADO`.
- **Explicit Device Identification / Requesting ID:** The master requests explicitly the device ID by setting the *ID Request* bit in the AL Control Register (0x0120.5) and the slave stores the DIV in the AL Status Code Register (0x0134). This is the preferred mechanism for complex slaves and is defined in [ETG.1000.6] and described in more detail in [ETG.1020]. The slave stack has to be compiled with *CFG_ESS_EXPLICIT_DEVICE_ID* set to 1 to enbale support for this identification method and the DIV has to be returned to the stack with the callback *cbStateRequest* in the structure *ESS_CBDATA_STATE_REQUEST*. The example provided in *.../apps/complex.c*. demonstrates the application usage if the stack contains the the support. This identification method has to be indicated to the master in the *Esi:Info:IdentificationReg134* element and/or the related SII element [ETG.2010].

2.3.3 PDO Mapping

The PDO mapping is defined or updated by passing arrays of the type `ESS_PDO_ENTRY` to *essODUpdatePDOConfiguration().*

The mapping of objects with a byte size of less or equal 31 bytes is performed with a single entry for each object to be mapped. For the example below let's assume we have the objects 0x2011:00 with 32 bits and 0x2012:00 with 8 bits which should be right after each other.

Example 1 (Standard mapping):

```
static const ESS_PDO_ENTRY PDO1a00Entries[] = {
      ESS_MAP_ENTRY(0x2011, 0, 32),
      ESS_MAP_ENTRY(0x2012, 0, 8 )
};
```

Sometimes it is necessary to insert padding data (dummy entries) for alignment reasons. For the example below let's assume we have the objects 0x2013:00 with 16 bits and 0x2014:00 with 32 bits and want to force the 2nd object to be on a 32 bit aligned address.

Example 2 (Dummy Mapping):

```
static const ESS_PDO_ENTRY PDO1a00Entries[] = {
      ESS_MAP_ENTRY(0x2013, 0, 16),
      ESS_MAP_DUMMY(16),            /* Align to 32 bits */
      ESS_MAP_ENTRY(0x2014, 0, 32 )
};
```

If the data size of the object exceeds 31 bytes the [ETG.1020] describes the mechanism to define the mapping of such objects. The mapping starts with a normal entry with a length of 30 bytes (240 bits) followed by *extension* entries of chunks with 30 bytes (240 bits) apart from the last chunk which might be smaller. The stack will internally apply the required copy operations for the extension entries. For the example below let's assume we have an object 0x2015:00 with 64 bytes.

Example 3 (Mapping of objects with more than 31 bytes):

```
static const ESS_PDO_ENTRY PDO1a00Entries[] = {
      ESS_MAP_ENTRY(0x2015, 0, 240), /* Map 30 bytes */
      ESS_MAP_EXTEND(240),           /* Map 30 bytes */
      ESS_MAP_EXTEND(32),            /* Map last 4 bytes */
};
```

2.3.4 Process Data Exchange

The application is responsible to send the current input data to the EtherCAT master and process the output data received from the EtherCAT master.

The local input handler which are responsible to send the current input data to the master should be active in the states SAFEOP and OP.

The local output handler which are responsible to set the current output data received from the master should only be active in the OP state. In the SAFEOP state the device/application specific safe-state values should be set.

The application should track the current state of the local input and output handler by attaching the *cbInOutputsActive()* callback handler (see 2.5). If the local output handler is inactive (as the device is in the SAFEOP state) valid output data might still be received by the EtherCAT master as the respective SM remain active and so the *cbOutputsUpdated()* callback are called. The application is responsible to set all variables to their application/device specific safe-state values instead of using the data received by the EtherCAT master.

2.4 Function description

Unless otherwise noticed, all parameters are mandatory, i.e. pointer values must not be *NULL*. For most pointers it's also noticed that "data must remain there": This means that data is not copied from pointer location and therefore the pointer must remain valid even after function call.

2.4.1 essGetVersion()

Returns the stack version. Can be called without stack being started.

```
const char* essGetVersion(void);
```

Return values

Pointer to a version string, e.g. *"Version 1.0.0, build Jun 16 2011 13:51:02"*.

2.4.2 essGetTime()

Returns a 32 bit time stamp. In milliseconds, wraps after *0xffffffff*.

```
ESS_TIMESTAMP essGetTime(void);
```

2.4.3 essFormatResult()

Returns an *ESS_RESULT* as string. Can be called without stack being started.

```
const char* essFormatResult(ESS_RESULT err);
```

err

Result to be formatted, see *ESS_RESULT*.

Return values

Pointer to a string that describes *err*, e.g. *"ESS_RESULT_SUCCESS"*.

2.4.4 essOpen()

Opens and initializes an EtherCAT device. (Please note “essOpen() / essClose() serialization” remark below)

```
ESS_RESULT essOpen(ESS_DEVICE_INDEX devIdx,
                   const ESS_CONFIGURATION* config,
                   ESS_HANDLE* resHandle);
```

devIdx

Index of EtherCAT device to be used, starting with index 0.

config

Pointer to configuration information, see *ESS_CONFIGURATION*. Data must remain there. (*NULL* is also allowed, but intended only for usage with *essIoctl()* – certain functions requiring a valid configuration, e.g. *essStart()*, are not possible then and will just return *ESS_RESULT_INVALID_CONFIG* then)

resHandle

Pointer to where the resulting *ESS_HANDLE* shall be stored.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_INVALID_INDEX* when the device does not exist, *ESS_RESULT_ALREADY_ACTIVE* when the device is already opened, *ESS_RESULT_MISSING_CALLBACK* when a mandatory callback in the given *ESS_CONFIGURATION* is *NULL*.

2.4.5 essClose()

Closes a device. Handle *h* becomes invalid then and must not be used any more. If the stack is still running, *essStop()* has to be called before this function can be called. (Please note “essOpen() / essClose() serialization” remark below)

```
ESS_RESULT essClose(ESS_HANDLE h);
```

h

Handle to the device, created with *essOpen()*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

essOpen() / essClose() serialization:
If multiple devices are used by multiple threads/tasks, then all calls to essOpen() and essClose() must be serialized, i.e. there must never be another thread/task at the same time in essOpen() or essClose().

2.4.6 essStart()

Starts the stack. On success the function does not return and the application runs only by stack’s callbacks then. To stop the stack *essStop()* has to be called within the cyclic callback.

```
ESS_RESULT essStart(ESS_HANDLE h);
```

h

Handle to the device, created with *essOpen()*.

Return values

On success (when stopped by *essStop()*) the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_CANT_OPEN* when the device could not be opened, e.g. on Windows when the device driver was not loaded/installed correctly.

2.4.7 essStop()

Stops the stack. Must be called from within the cyclic callback.

```
ESS_RESULT essStop(ESS_HANDLE h);
```

h

Handle to the device, created with *essOpen()*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.8 essIndicateError()

Sets new EtherCAT state and error status. Usually used when an error is detected outside a state transition and to lower the EtherCAT state – e.g. from “Op” to “SafeOp” when the outputs can’t be written any more, etc.

The *newState* parameter is checked and fixed/ignored when wrong, i.e. the function will still return *ESS_RESULT_SUCCESS* then.

Not all invalid transitions can be checked, make sure desired action is allowed. (Check ETG documents and use the CTT)

```
ESS_RESULT essIndicateError(ESS_HANDLE h,
                            ESC_STATE newState,
                            REG_VAL_ALSTATUSCODE errCode);
```

h

Handle to the device, created with *essOpen()*.

newState

New EtherCAT state for the slave.

errCode

Error code to write to the *AL Status Code* register (0x0134:0x0135).

See *REG_VAL_ALSTATUSCODE* for constants to use.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.9 essCoESendEmergency()

Sends a CoE emergency message.

```
ESS_RESULT essCoESendEmergency(ESS_HANDLE h, const ESS_COE_EMERGENCY* data);
```

h

Handle to the device, created with *essOpen()*.

data

Pointer to message details, see *ESS_COE_EMERGENCY*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_TRY_AGAIN* when there’s already an unsent CoE emergency. *ESS_RESULT_INVALID_CONFIG* when no mailboxes are configured. *ESS_RESULT_WRONG_ECAT_STATE* when mailboxes are not set up (e.g. in Init state).

2.4.10 essSyncInputs()

Informs the stack that entries that might be mapped into input process data have been modified by the application, i.e. the stack shall update the according SM buffer.

Usually this is called with *ESS_ISYN_FLAGS_NONE*: that will schedule the update of all input SMs until callback is done or next loop – thus allowing multiple essSyncInputs() calls without affecting performance.

Alternatively an immediate update of a specific SM may be forced, e.g. with flags set to *ESS_ISYN_FLAGS_SM3_IMMEDIATELY* for SM3. This should be used only if the input data must be available for EtherCAT read as fast as possible – it may seriously affect performance if called more often than required (mind the ESC connection, e.g. SPI access).

```
ESS_RESULT essSyncInputs(ESS_HANDLE h, ESS_ISYN_FLAGS flags);
```

h

Handle to the device, created with *essOpen()*.

flags

ESS_ISYN_FLAGS_NONE or e.g. *ESS_ISYN_FLAGS_SM3_IMMEDIATELY*, as described above.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when slave is not in SafeOp/Op state.

2.4.11 essEEPROMRead()

Reads data from slave's EEPROM. Without any swapping.

Address and length are not checked for being outside actual EEPROM size – read bytes are just invalid (e.g. mirrored from another address) then.

```
ESS_RESULT essEEPROMRead(ESS_HANDLE h,
                         uint32_t addr,
                         uint32_t len,
                         void* dest);
```

h

Handle to the device, created with *essOpen()*.

addr

Byte offset where to read from. Must be even.

len

Byte length to read. Must be greater than 0.

dest

Where to store the read data.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_CANT_OPEN* when EtherCAT currently has access to EEPROM (determined by ESC register 0x0500).

2.4.12 essEEPROMWrite()

Writes data to slave's EEPROM. Without any swapping.

Address and length are not checked for being outside actual EEPROM size – additional bytes might be written to another address!

```
ESS_RESULT essEEPROMWrite(ESS_HANDLE h,
                          uint32_t addr,
                          uint32_t len,
                          const void* src);
```

h

Handle to the device, created with *essOpen()*.

addr

Byte offset where to write to. Must be even.

len

Byte length to write. Must be greater than 0 and even.

src

Where to read the data from.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_CANT_OPEN* when EtherCAT currently has access to EEPROM (determined by ESC register: 0x0500).

2.4.13 essSetLEDState()

Sets the status of a LED. Requires HAL/driver to have this implemented – see information for architecture/platform you are using. (*ESC_LED_STATE_ON* turns the LED on immediately, all other states turn it off immediately and are updated by the stack loop)

```
ESS_RESULT essSetLEDState(ESS_HANDLE h,
                          ESC_LED_TYPE led,
                          ESC_LED_STATE state);
```

h

Handle to the device, created with *essOpen()*.

led

Which LED to change, see *ESC_LED_TYPE*.

state

New state of that LED, see *ESC_LED_STATE*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_INVALID_ARG* when the selected LED is not supported.

2.4.14 essESCRead()

Reads a custom number of bytes from ESC memory.

```
void essESCRead(ESS_HANDLE h, uint16_t address, void* dest, uint16_t len);
```

h

Handle to the device, created with *essOpen()*.

address

ESC source address.

dest

Pointer to where to store the read data.

len

Number of bytes to read.

2.4.15 essESCWrite()

Writes a custom number of bytes to ESC memory.

```
void essESCWrite(ESS_HANDLE h, uint16_t address,
                 const void* src, uint16_t len);
```

h

Handle to the device, created with *essOpen()*.

address

ESC destination address.

src

Pointer to where to read from.

len

Number of bytes to write.

2.4.16 essESCRead8()

Reads a Byte from ESC memory.

```
uint8_t essESCRead8(ESS_HANDLE h, uint16_t address);
```

h

Handle to the device, created with *essOpen()*.

address

ESC address.

2.4.17 essESCRead16()

Reads a Word from ESC memory. Value will be swapped automatically on big endian systems.

```
uint16_t essESCRead16(ESS_HANDLE h, uint16_t address);
```

h

Handle to the device, created with *essOpen()*.

address

ESC address.

2.4.18 essESCRead32()

Reads a DWord from ESC memory. Value will be swapped automatically on big endian systems.

```
uint32_t essESCRead32(ESS_HANDLE h, uint16_t address);
```

h

Handle to the device, created with *essOpen()*.

address

ESC address.

2.4.19 essESCWrite8()

Writes a Byte to ESC memory.

```
void essESCWrite8(ESS_HANDLE h, uint16_t address, uint8_t value);
```

h

Handle to the device, created with *essOpen()*.

address

ESC Address.

value

Value to write.

2.4.20 essESCWrite16()

Writes a Word to ESC memory. Value will be swapped automatically on big endian systems.

```
void essESCWrite16(ESS_HANDLE h, uint16_t address, uint16_t value);
```

h

Handle to the device, created with *essOpen()*.

address

ESC Address.

value

Value to write.

2.4.21 essESCWrite32()

Writes a DWord to ESC memory. Value will be swapped automatically on big endian systems.

```
void essESCWrite32(ESS_HANDLE h, uint16_t address, uint32_t value);
```

h

Handle to the device, created with *essOpen()*.

address

ESC Address.

value

Value to write.

2.4.22 essEoESendFrame()

Sends an Ethernet frame to EtherCAT. Only implemented when stack was built with EoE support (*CFG_ESS_SUPPORT_EOE* in *essConfig.h*)

Does not check the EtherCAT state: when mailboxes are set up correctly the data is sent: it's up to the application to determine whether this is currently allowed.

```
ESS_RESULT essEoESendFrame(ESS_HANDLE h,
                           const void* data,
                           uint16_t dataLen,
                           void* reserved);
```

h

Handle to the device, created with *essOpen()*.

data

Pointer to the frame data.

dataLen

Length of frame data in byte.

reserved

Must be *NULL*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_TRY_AGAIN* when there's still a frame (or fragment of it) to be sent (Sample application *EoE.c* shows how to implement a simple queue).

2.4.23 essGetTag()

Returns the user defined pointer that was set in *ESS_CONFIGURATION*'s tag member.

```
void* essGetTag(ESS_HANDLE h);
```

h

Handle to the device, created with *essOpen()*.

2.4.24 essIoctl()

Used to implement other hardware/platform specific functions. Shouldn't be needed for regular use: Most IOCTLs are not implemented on most hardware.

With the Source Code Version this can be used to implement own functions / to communicate with HAL from application.

```
ESS_RESULT essIoctl(ESS_HANDLE h, ESS_IOCTL fn, void* data,
                    uint32_t sizeOfData);
```

h

Handle to the device, created with *essOpen()*.

fn

The function that shall be performed. (See *ESS_IOCTL_...* #defines in *essTypes.h*: own functions shall start at *0x80000000*).

data

Pointer to function specific data

sizeOfData

Number of bytes usable at *data*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_NOT_IMPLEMENTED* if function is not supported/ implemented. *ESS_RESULT_INSUFFICIENT_BUFFER* if *sizeOfData* is not sufficient to fulfill the requested function – try again with increased buffer then.

2.4.25 CoE object dictionary specific

2.4.25.1 essODCreate()

Creates the object dictionary for the slave. Only possible in Init state.

```
ESS_RESULT essODCreate(ESS_HANDLE h, ESS_OD_FLAGS flags);
```

h

Handle to the device, created with *essOpen()*.

flags

Flags for dictionary creation, see *ESS_OD_FLAGS*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when not in Init state. *ESS_RESULT_ALREADY_ACTIVE* when already created.

2.4.25.2 essODDelete()

Deletes the object dictionary for the slave (and all its objects/entries). Only possible in Init state.

```
ESS_RESULT essODDelete(ESS_HANDLE h, uint16_t reserved);
```

h

Handle to the device, created with *essOpen()*.

reserved

For future use, must be set to 0.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when not in Init state.

(Returns *ESS_RESULT_SUCCESS* also if dictionary was already deleted)

2.4.25.3 essODObjectAdd()

Adds an object to the dictionary. Only possible after dictionary was created and only in Init/PreOp state.

```
ESS_RESULT essODObjectAdd(ESS_HANDLE h,
                          uint16_t idx,
                          ESS_OD_OBJECT_FLAGS flags,
                          const ESS_OD_OBJECT_INFOS* infos);
```

h

Handle to the device, created with *essOpen()*.

idx

Index of the object to be added.

flags

Creation flags for the object, see *ESS_OD_OBJECT_FLAGS*.

infos

Pointer to *ESS_OD_OBJECT_INFOS* for this object, or *NULL* when “SDO Information Service” is not needed.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when slave is not in Init/PreOp state. *ESS_RESULT_OBJ_EXISTS* when an object with that index already exists.

2.4.25.4 essODObjectDelete()

Deletes an object (and all its entries) from the dictionary. Only possible in Init/PreOp state.

```
ESS_RESULT essODObjectDelete(ESS_HANDLE h, uint16_t idx);
```

h

Handle to the device, created with *essOpen()*.

idx

Index of the object.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when slave is not in Init/PreOp state. *ESS_RESULT_OBJ_NOT_FOUND* when object does not exist.

2.4.25.5 essODEntryAdd()

Adds an entry to an existing object in the dictionary. Only possible in Init/PreOp state.

```
ESS_RESULT essODEntryAdd(ESS_HANDLE h,
                         uint16_t idx,
                         uint8_t subIdx,
                         uint32_t bitLen,
                         void* data,
                         COE_ACCESS access,
                         ESS_OD_ENTRY_FLAGS flags,
                         const ESS_OD_ENTRY_INFOS* infos);
```

h

Handle to the device, created with *essOpen()*.

idx

Index of object this entry belongs to.

subIdx

Entry index, 0 to 255.

bitLen

Bit length of the entry.

data

Pointer to the entry data. (Must be little endian. *ess.h* offers some macros to be endianness independent, e.g. *SWAP32_HOST2LE()* and *SWAP32_LE2HOST()*)

NULL when data shall be handled by *ESS_CB_COE_READWRITE*.

access

Access rights to the entry, see *COE_ACCESS*.

flags

Entry creation flags, see *ESS_OD_ENTRY_FLAGS*.

infos

Pointer to *ESS_OD_ENTRY_INFOS* for this entry, or *NULL* when “SDO Information Service” is not needed.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_ENTRY_EXISTS* when an entry with that sub index already exists. *ESS_RESULT_OBJ_NOT_FOUND* when the object does not exist.

2.4.25.6 essODEntryDelete()

Deletes an entry from an object in the dictionary. Only possible in Init/PreOp state.

```
ESS_RESULT essODEntryDelete(ESS_HANDLE h,
                            uint16_t idx,
                            uint8_t subIdx);
```

h

Handle to the device, created with *essOpen()*.

idx

Object index.

subIdx

Entry index.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_WRONG_ECAT_STATE* when slave is not in Init/PreOp state. *ESS_RESULT_OBJ_NOT_FOUND* when the object does not exist.

ESS_RESULT_ENTRY_NOT_FOUND when the entry does not exist.

2.4.25.7 essODAddArrayObject()

Adds an array object to the dictionary (COE_CODE = *COE_CODE_ARRAY*). All subentries are automatically created. *entriesCount* determines the number of array elements, i.e. with an *entriesCount* value of *5* for example, there will be six entries: Subindex 0 with a value of 5 (“Max Subindex”), and 5 more sub entries which represent the array elements.

Could be replaced by *essODObjectAdd()* and several *essODEntryAdd()* calls, of course, but by this function internally only one entry is created – thus saving a lot of RAM when many entries are required. (But currently only usable for arrays of fix length)

```
ESS_RESULT essODAddArrayObject(ESS_HANDLE h,
                               uint16_t idx,
                               const char* name,
                               uint8_t entriesCount,
                               COE_DATATYPE entriesDataType,
                               uint32_t entryBitLen,
                               COE_ACCESS entriesAcc,
                               void* entriesData,
                               uint16_t entriesDataStride);
```

h

Handle to the device, created with *essOpen()*.

idx

Index of the object.

name

Name of the object, for “SDO Information Service”

entriesCount

Number of array elements. The first entry, with subindex 0, will be an read-only entry with this value.

entriesDataType

COE_DATATYPE for the entries, needed for “SDO Information Service”

entryBitLen

Bit length of each item.

entriesAcc

Access rights for each item, see *COE_ACCESS*.

entriesData

Pointer to the data of the first array element (with subindex 1) or *NULL*, when data shall be handled by *ESS_CB_COE_READWRITE*.

entriesDataStride

Ignored when *data* is *NULL*. Else distance between the array elements in byte, i.e. when *entriesData* points to a packed array of *uint32_t* values for example, then *entriesDataStride* must be *4*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.8 essODAddGenericObjects()

Adds some common objects to the dictionary. Only possible after dictionary was created and only in Init/PreOp state.

This function is just a convenience wrapper that could be replaced by a few *essODObjectAdd()* and *essODEntryAdd()* calls – but as some of these objects are mandatory for complex slaves, it's at least useful to save some lines of application code.

```
ESS_RESULT essODAddGenericObjects(ESS_HANDLE h,
                                  const uint32_t* deviceType,
                                  const uint32_t* vendorId,
                                  const uint32_t* productCode,
                                  const uint32_t* revNo,
                                  const uint32_t* serialNo,
                                  const char* devName,
                                  const char* verHardware,
                                  const char* verSoftware);
```

h

Handle to the device, created with *essOpen()*.

deviceType

Pointer to device type integer (Object 0x1000). Data must remain there and must be in little-endian format.

vendorId

Pointer to vendor id integer (Object 0x1018). Data must remain there and must be in little-endian format.

productCode

Pointer to product code integer (Object 0x1018). Data must remain there and must be in little-endian format.

revNo

Pointer to revision number integer (Object 0x1018). Data must remain there and must be in little-endian format.

serialNo

Pointer to serial number (Object 0x1018). Data must remain there and must be in little-endian format.

devName

Optional pointer to device name string (Object 0x1008). Data must remain there.

verHardware

Optional pointer to hardware version string (Object 0x1009). Data must remain there.

verSoftware

Optional pointer to software version string (Object 0x100a). Data must remain there.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Common failures: *ESS_RESULT_INVALID_ARG* when a non optional parameter was *NULL*.

2.4.25.9 essODUpdatePDOConfiguration()

Sets the configuration of a PDO (also called mapping / defines which entries are in that PDO). Only possible in Init/PreOp.

```
ESS_RESULT essODUpdatePDOConfiguration(ESS_HANDLE h,
                                       uint16_t pdo,
                                       const char* name,
                                       const ESS_PDO_ENTRY* entries,
                                       uint8_t entryCount,
                                       ESS_BOOL writable);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of PDO object. (0x1600..0x17ff: Rx PDOs, 0x1a00..0x1bff: Tx PDOs)

name

Pointer to the name of the object. Data must remain there.

entries

Pointer to the entries within that PDO, see ESS_PDO_ENTRY.

When *NULL*: PDO Object shall be deleted from dictionary.

entryCount

Number of items at *entries*.

writable

When set to *ESS_TRUE* the object becomes writable (in PreOp), thus allowing a dynamic PDO configuration.

To obtain the current PDO configurations use *essODGetPDOConfiguration()*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.10 essODGetPDOConfiguration()

Used to retrieve the current content of a PDO.

```
ESS_RESULT essODGetPDOConfiguration(ESS_HANDLE h,
                                    uint16_t pdo,
                                    uint8_t* numEntries,
                                    ESS_PDO_ENTRY* dest,
                                    uint8_t reserved);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

numEntries

When *dest* is *NULL* the number of items in the PDO is written to the value at *numEntries*. Else the value at *numEntries* is used to determine the maximum number of items to store at *dest* and to return the actual number of items written to *dest* afterwards.

dest

Pointer to where to store the resulting items. When *NULL* only the number of items in the PDO is returned (by *numEntries*).

reserved

For future use, must be set to 0.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.11 essODUpdatePDOAssignment()

Sets the slave's PDO assignment. Only possible in Init/PreOp state. Has to be called for every SM, even if now PDOs are assigned by default.

```
ESS_RESULT essODSetPDOAssignment(ESS_HANDLE h,
                                 ESS_SM smIdx,
                                 const uint16_t* pdo,
                                 uint8_t pdoCount,
                                 ESS_BOOL writable);
```

h

Handle to the device, created with *essOpen()*.

smIdx

SM whose assignment is to be updated.

Most common configuration: Outputs/RxPDOs: *ESS_SM_2*, Inputs/TxPDOs: *ESS_SM_3*.

pdo

Pointer to *uint16_t* array. *NULL* if no PDOs are assigned *pdoCount* is also *0*.

pdoCount

Number of items at *pdo*.

writable

When set to *ESS_TRUE* the object becomes writable (in PreOp), thus allowing a dynamic PDO assignment.

To obtain the current PDO assignments use *essODGetPDOAssignment()*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.12 essODGetPDOAssignment()

Used to retrieve the PDOs assigned to a specific SM.

```
ESS_RESULT essODGetPDOAssignment(ESS_HANDLE h,
                                 ESS_SM smIdx,
                                 uint8_t* numPDOs,
                                 uint16_t* dest);
```

h

Handle to the device, created with *essOpen()*.

smIdx

SM index. (Common configuration: *ESS_SM_3* for inputs and *ESS_SM_2* for outputs)

numPDOs

When *dest* is *NULL* the number PDOs assigned to that SM is written to the value at *numPDOs*. Else the value at *numPDOs* is used to determine the maximum number of items to store at *dest* and to return the actual number of items written to *dest* afterwards.

dest

Pointer to where to store the resulting items. When *NULL* only the number of PDOs assigned to that SM is returned (by *numPDOs*).

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.13 essODPDOParamCreate()

Creates a parameter object for a PDO, according to [ETG.1020]. (Object 0x1800 for PDO 0x1a00, etc., requires *CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS*)

See following functions for how to handle the parameter data. The stack does not use any of the parameter data itself – the PDO parameter objects are used by the master or configuration tool etc.

(This function is just a convenience function that could be replaced by *essODObjectAdd()*/ *essODEntryAdd()* calls and application handling of the entry data)

```
ESS_RESULT essODPDOParamCreate(ESS_HANDLE h,
                               uint16_t pdo,
                               const char* name,
                               ESS_OD_PDOPARAM_FLAGS flags);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

name

Name of the object for SDO information service, e.g. *"RxPDO Parameter"*.

flags

Used to configure the entries, see *ESS_OD_PDOPARAM_FLAGS*.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

2.4.25.14 essODPDOParamUpdateExclude()

Used to update the list of excluded PDOs, i.e. PDOs that must not be assigned at the same time. (Parameter object subindex 6)

(Requires *CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS*, usable only on objects created with *essODPDOParamCreate()*.)

```
ESS_RESULT essODPDOParamUpdateExclude(ESS_HANDLE h,
                                      uint16_t pdo,
                                      const void* data,
                                      uint16_t dataLen);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

data

Pointer to an array of PDO indices to exclude. Indices must be little-endian.

dataLen

Date length in bytes (must be an even value).

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Example: (PDO 0x1a00 shall exclude PDO 0x1a01 and PDO 0x1a02)

```
res = essODPDOParamUpdateExclude(hDev, 0x1a00, "\x01\x1a\x02\x1a", 4);
```

2.4.25.15 essODPDOParamGetState()

Used set a pointer to the state value. (Parameter object subindex 7)

(Requires *CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS*, usable only on objects created with *essODPDOParamCreate()*.)

```
ESS_RESULT essODPDOParamGetState(ESS_HANDLE h,
                                 uint16_t pdo,
                                 ESS_BOOL** state);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

state

Pointer to the state value pointer.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

See *essODPDOParamGetToggle()* for an example.

2.4.25.16 essODPDOParamGetControl()

Used to set a to pointer to the control value. (Parameter object subindex 8)

(Requires *CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS*, usable only on objects created with *essODPDOParamCreate()*.)

```
ESS_RESULT essODPDOParamGetControl(ESS_HANDLE h,
                                   uint16_t pdo,
                                   ESS_BOOL** control);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

control

Pointer to the control value pointer.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

See *essODPDOParamGetToggle()* for an example.

2.4.25.17 essODPDOParamGetToggle()

Used to set a a pointer to the toggle value. (Parameter object subindex 9)

(Requires *CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS*, usable only on objects created with *essODPDOParamCreate()*.)

```
ESS_RESULT essODPDOParamGetToggle(ESS_HANDLE h,
                                  uint16_t pdo,
                                  ESS_BOOL** toggle);
```

h

Handle to the device, created with *essOpen()*.

pdo

Index of the PDO object, e.g. 0x1600/0x1a00.

toggle

Pointer to the toggle value pointer.

Return values

On success the function returns *ESS_RESULT_SUCCESS*.

Example: (analogue for control and state value)

```
ESS_BOOL* rxToggle;
ESS_BOOL* txToggle;

essODPDOParamGetToggle(hDev, 0x1600, &rxToggle);
essODPDOParamGetToggle(hDev, 0x1a00, &txToggle);
/* Return values must be checked in your code! */

/* After Outputs were updated: */
if (*rxToggle)
   OutputsUpdated();

/* e.g. in cyclic handler: */
if (InputsUpdated())
  *txToggle = !*txToggle;
```

2.5 Callbacks

All callbacks are set with the *ESS_CONFIGURATION/ESS_CALLBACKS* struct, given to the stack by *essOpen()*. Unless otherwise described here, every callback is mandatory (i.e. its member variable must point to a valid function).

All data that is accessible via the callback parameter is read only, unless otherwise described in its type description in section 2.6.

Type definition **Parameter** **Member name**	**Description**
ESS_CB_CYCLIC *ESS_CBDATA_CYCLIC** *cbCyclic*	Called cyclically – this should be the application's main loop. The interval is set by *halConfig* member of *ESS_CONFIGURATION* struct.
ESS_CB_STATE_REQUEST *ESS_CBDATA_STATE_REQUEST** *cbStateRequest*	Called when EtherCAT requests a new state for the slave or requests the configured device ID via the 'Explicit Device ID' mechanism described in [ETG.1020].
ESS_CB_SYNCMANAGER *ESS_CBDATA_SM** *cbSMInterrupt*	Called when the AL Event register (0x0220:0x0223) signals an SM interrupt. Might be used to "hook" into these interrupts – but replacing that interrupt handling by application would lead to almost writing an own EtherCAT stack. Used with *CFG_ESS_SM_CALLBACK*.
ESS_CB_INOUTPUTS_ACTIVATE *ESS_CBDATA_INOUTPUTS_ACTIVATE** *cbInOutputsActivate*	Called when the local input/output handler shall be started or stopped (see 2.3.4). Determined just by state changes – e.g. called for outputs even when no outputs exist/mapped.
ESS_CB_OUTPUTS_UPDATED *ESS_CBDATA_SM_EVENT** *cbOutputsUpdated*	Optional. Called when an SM buffer (for an SM that is configured as "Outputs") was written by EtherCAT.
ESS_CB_INPUTS_UPDATED *ESS_CBDATA_SM_EVENT** *cbInputsUpdated*	Optional. Called when an SM buffer (for an SM that is configured as "Inputs") was read by EtherCAT. That interrupt is only cleared when the application writes new inputs to the SM buffer, i.e. when *essSyncInputs()* is used – so you shouldn't use this callback if the inputs are not updated in every cycle. When this callback is not used, you should also mask that interrupt to save CPU load (Clear the inputs SM bit in *ESC_REG_ALEVENTMASK*).
ESS_CB_COE_READWRITE *ESS_CBDATA_COE_READWRITE** *cbCoEReadWrite*	Optional (But required when CoE items without data pointer exist, see *essODEntryAdd()*). Called whenever the stack has or needs data for an entry. Used with *CFG_ESS_SUPPORT_COE*. **Note:** The member variable *abortCode* of *ESS_CBDATA_COE_READWRITE* is set to the default value COE_ABORTCODE_SUBINDEX by the stack. If the application wants the CoE request to succeed it has to be changed explicitly to COE_ABORTCODE_NONE.

Type definition Parameter Member name	Description
ESS_CB_COE_EVENT *ESS_CBDATA_COE_EVENT** *cbCoEEvent*	Optional. Called for certain CoE events, see *ESS_OD_ENTRY_CALLBACK* for available callbacks. Used with *CFG_ESS_SUPPORT_COE*. **Note:** The member variable *abortCode* of *ESS_CB_COE_EVENT* is set to the default value COE_ABORTCODE_NONE by the stack.
ESS_CB_FOE_OPEN *ESS_CBDATA_FOE_OPEN** *cbFoEOpen*	Called when an FoE upload or download is started. Optional, but when used both other FoE callbacks must be set, too. Used with *CFG_ESS_SUPPORT_FOE*.
ESS_CB_FOE_CLOSE *ESS_CBDATA_FOE_CLOSE** *cbFoEClose*	Called when an FoE transfer is finished. Optional, but when used both other FoE callbacks must be set, too. Used with *CFG_ESS_SUPPORT_FOE*.
ESS_CB_FOE_DATA *ESS_CBDATA_FOE_DATA** *cbFoEData*	Called when data for a FoE transfer is available or required. Optional, but when used both other FoE callbacks must be set, too. Used with *CFG_ESS_SUPPORT_FOE*.
ESS_CB_EOE_SETIPPARAM *ESS_CBDATA_EOE_SETIPPARAM** *cbEoESetIPParam*	Optional. Called when EoE packet to set IP parameters was received. Used with *CFG_ESS_SUPPORT_EOE*.
ESS_CB_EOE_SETADDRFILTER *ESS_CBDATA_EOE_SETADDRFILTER** *cbEoESetAddrFilter*	Called when EoE packet to set IP address filters was received. Used with *CFG_ESS_SUPPORT_EOE*.
ESS_CB_EOE_FRAME *ESS_CBDATA_EOE_FRAME** *cbEoEFrame*	Called when an Ethernet frame was received via EoE. Used with *CFG_ESS_SUPPORT_EOE*.
ESS_CB_VOE *ESS_CBDATA_VOE** *cbVoE*	Optional. Called when a VoE packet was received. Used with *CFG_ESS_SUPPORT_VOE*.
ESS_CB_EEPROM_EMULATION *ESS_CBDATA_EEPROM_EMULATION** *cbEEPROMEmulation*	Optional. Called for each EEPROM access when ESC does not handle EEPROM itself. Used with *CFG_ESS_EEPROM_EMULATION*.
ESS_CB_DC *ESS_CBDATA_DC** *cbDCEvent*	Called for DC SYNC/Latch events, i.e. when bit 1..3 in ESC reg. 0x0220 is set. (Useful only with acknowledge mode for SYNC event) Used with *CFG_ESS_DC_CALLBACK*.
ESS_CB_AOE *ESS_CBDATA_AOE** *cbAoE*	Optional. Called when a AoE packet was received. Used with *CFG_ESS_SUPPORT_AOE*.
ESS_CB_SOE *ESS_CBDATA_SOE** *cbSoE*	Optional. Called when a SoE packet was received. Used with *CFG_ESS_SUPPORT_SOE*.

2.6 Data types

All listed types are defined in *essTypes.h* and *ecatDefs.h* (both included by *ess.h*).

2.6.1 ESS_HANDLE

Data behind this is not accessible by application. It's just created by *essOpen()* and then used for calling other functions.

Remarks:

> Calling a function with an invalid handle (one that was not obtained by a successful call to *essOpen()* or one that became invalid by calling *essClose()*) can have any undesired side effect – at least crashing the application.

2.6.2 ESS_BOOL

#define	Description
ESS_FALSE	0
ESS_TRUE	All other values

Remarks:

> Never compare with *ESS_TRUE*, i.e. do NOT test: "*if (x == ESS_TRUE)*".
>
> Use "*if (x != ESS_FALSE)*" instead. (Or just "*if (x)*")

2.6.3 ESS_RESULT

Returned by almost every stack function. Use *essFormatResult()* to get it as string.

#define	Description
ESS_RESULT_SUCCESS	No error. The function call succeeded.
ESS_RESULT_UNKNOWN	An unspecified error occurred. Check debug outputs with debug version for more details.
ESS_RESULT_INVALID_INDEX	An index was invalid, e.g. a *devIdx* parameter is out of range.
ESS_RESULT_NOT_READY	Something is not ready, e.g. EEPROM when ESC registers signal it has not been loaded.
ESS_RESULT_ALREADY_ACTIVE	Something was already done, e.g. device with that index was already opened.
ESS_RESULT_CANT_OPEN	Usually when device could not be opened, e.g. because driver was not installed/started correctly.
ESS_RESULT_TRY_AGAIN	Some action is temporarily not possible, e.g. sending an EoE frame or CoE emergency when there's already one to be sent.
ESS_RESULT_MISSING_CALLBACK	A mandatory callback of *ESS_CONFIGURATION/ESS_CALLBACKS* was NULL when *essOpen()* was called.
ESS_RESULT_INVALID_HANDLE	A handle given to a stack function is invalid. Usually only tested in debug versions – so don't rely on this

#define	Description
	and make sure no invalid handles are used.
ESS_RESULT_INVALID_LIST	Currently unused.
ESS_RESULT_OBJ_NOT_FOUND	Used in OD functions when object with given index does not exist.
ESS_RESULT_ENTRY_NOT_FOUND	Used in OD functions when entry with given sub index does not exist in that object.
ESS_RESULT_NO_DICT	Used in OD functions, e.g. when trying to add objects before the dictionary was created.
ESS_RESULT_NO_MEM	Out of memory. Usually in OD functions.
ESS_RESULT_OBJ_EXISTS	Used when a CoE object already existed that should have been added.
ESS_RESULT_ENTRY_EXISTS	Used when a CoE object entry already existed that should have been added.
ESS_RESULT_WRONG_ECAT_STATE	An action is not possible in the current EtherCAT state, e.g. changing the CoE dictionary when not in Init/PreOp state.
ESS_RESULT_INVALID_CONFIG	A configuration is wrong, e.g. when trying to do mailbox operations but the slave has no mailboxes configured. Also used when configuration given to *essOpen()* is invalid.
ESS_RESULT_INVALID_OD_FLAGS	Used when an action requires that the CoE dictionary was created with other flags.
ESS_RESULT_INVALID_SUBINDEX	Used when CoE entry sub index is negative or greater than 255.
ESS_RESULT_INVALID_SM	Used when SM is invalid or does not exist, e.g. when setting SMs PDO assignment.
ESS_RESULT_INVALID_ARG	A function argument is invalid, e.g. *NULL* or out of range.
ESS_RESULT_NOT_IMPLEMENTED	The function is not implemented, e.g. EoE or FoE functions when the stack was built with *CFG_ESS_SUPPORT_EOE/CFG_ESS_SUPPORT_FOE* defined to *0*.
ESS_RESULT_INVALID_SIZE	Currently used when trying to send an EoE frame that is too large.
ESS_RESULT_BUSY	Used e.g. in EEPROM functions when EEPROM busy bit is not cleared within expected time.
ESS_RESULT_INVALID_PDI_CONTROL	Returned by *essOpen()* when the PDI Control register (0x0140:0x0141) signals that PDI is disabled or that device is in “Device Emulation” mode. (Slave’s EEPROM configuration area needs to be updated then – or was read incorrectly)
ESS_RESULT_ABI_INCOMPATIBLE	Returned by *essOpen()* when the *essABIVersion* member of the configuration states that the application was built with an incompatible header – update the header file and adapt your application to the changes.

#define	Description
ESS_RESULT_INVALID_TIMER_CONFIG	Returned by *essOpen()* when the *halConfig* member of the configuration contains a value for the cyclic timer that cannot be reached by the HAL/driver – usually a higher value is necessary then.
ESS_RESULT_MISSING_ACK	Currently used only in EEPROM functions when EEPROM interface did not signal ACK within expected time.
ESS_RESULT_INVALID_OBJECT	Returned by some OD functions when it's not possible with that object, e.g. for objects that are created automatically/implicit.
ESS_RESULT_INSUFFICIENT_BUFFER	Returned by *essIoctl()* if given data length is not sufficient to fulfill the requested function etc.

2.6.4 ESS_TIMESTAMP

Integral value, in milliseconds.

2.6.5 ESS_DEVICE_INDEX

Integral value from 0 to 255.

2.6.6 ESC_LED_TYPE

#define	Description
ESC_LED_TYPE_ERR	EtherCAT Error LED. (Usually red)
ESC_LED_TYPE_RUN	EtherCAT RUN LED. (Usually green)
*ESC_LED_TYPE_CUSTOM*n	Custom LEDs. Not used by stack, but HAL might offer these for application usage.

2.6.7 ESC_LED_STATE

As defined by ETG.1300 documents.

#define	Description
ESC_LED_STATE_OFF	LED is off.
ESC_LED_STATE_FLASH_1	LED repeats: 200 ms on, 1000 ms off.
ESC_LED_STATE_FLASH_2	LED repeats: 200 ms on, 200 ms off, 200 ms on, 1000 ms off.
ESC_LED_STATE_FLASH_3	LED repeats: 2x(200 ms on, 200 ms off), 200 ms on, 1000 ms off.
ESC_LED_STATE_FLASH_4	LED repeats: 3x(200 ms on, 200 ms off), 200 ms on, 1000 ms off.
ESC_LED_STATE_BLINK	LED repeats: 200 ms on, 200 ms off.
ESC_LED_STATE_FLICKER	LED repeats: 50 ms on, 50 ms off.
ESC_LED_STATE_ON	LED is on.

2.6.8 ESS_OD_FLAGS

#define	Description
ESS_OD_FLAGS_NONE	Dictionary is created with no special flags.
ESS_OD_FLAGS_HANDLE_SM_TYPES	Stack automatically creates the CoE object 0x1c00 (“Sm types”) according to SM configuration given in *essOpen()*.

2.6.9 ESS_OD_OBJECT_FLAGS

Currently no special flags exist.

#define	Description
ESS_OD_OBJECT_FLAGS_NONE	Object is created without special behavior.

2.6.10 ESS_OD_ENTRY_FLAGS

Currently no special flags exist.

#define	Description
ESS_OD_ENTRY_FLAGS_NONE	Entry is created without any special behavior.
ESS_OD_ENTRY_FLAGS_UPLOAD_TRIMMED	Trim trailing zeros for PDO upload. Only for entries without data callback. Usually used for strings.
ESS_OD_ENTRY_FLAGS_USE_DATA_AS_MAXSUBIDX	Data in this entry shall be used as max subindex for that object. Used to hide inactive entries from SDO information service.
ESS_OD_ENTRY_FLAGS_NOSDOINFO	Completely hide this entry from SDO information service.
ESS_OD_ENTRY_FLAGS_ALLOWPARTIALDOWNLOAD	If set, CoE download with less data than object’s data size is possible. Missing bytes (trailing bytes) will be filled with *0x00*. Flag is automatically set for *COE_DATATYPE_OCTETSTRING*, *COE_DATATYPE_WSTRING* and *COE_DATATYPE_STRING*.

2.6.11 ESS_OD_PDOPARAM_FLAGS

See [ETG.1020] for PDO parameter object details.

#define	Description
ESS_OD_PDOPARAM_FLAGS_NONE	Entry is created without any special behavior.
ESS_OD_PDOPARAM_FLAGS_EXCLPDOS_EXISTS	Entry with subindex 6 will be created. Handled by *essODPDOParamUpdateExclude()*.
ESS_OD_PDOPARAM_FLAGS_STATE_EXISTS	Entry with subindex 7 will be created. Handled by *essODPDOParamGetState()*.
ESS_OD_PDOPARAM_FLAGS_CONTROL_EXISTS	Entry with subindex 8 will be created. (Used only for Rx-PDOs) Handled by *essODPDOParamGetControl()*.
ESS_OD_PDOPARAM_FLAGS_TOGGLE_EXISTS	Entry with subindex 9 will be created. Handled by *essODPDOParamGetToggle()*.

2.6.12 ESS_OD_ENTRY_CALLBACK

#define ESS_OD_ENTRY_CALLBACK_	Description
REQUESTED_UPLOAD	Called when an SDO upload is requested, before testing if access is allowed or entry exists, etc.
REQUESTED_DOWNLOAD	Called when an SDO download is requested, before testing if access is allowed or entry exists, etc.
STARTING_UPLOAD	Called when an SDO upload is actually started, i.e. entry exists and access is allowed, etc.
STARTING_DOWNLOAD	Called when an SDO download is actually started, i.e. entry exists and access is allowed, etc.
COMPLETED_UPLOAD	Called when an SDO upload was finished successfully.
COMPLETED_DOWNLOAD	Called when an SDO download was finished successfully.
ABORTED_UPLOAD	Called when an SDO upload was aborted. (segmented transfer)
ABORTED_DOWNLOAD	Called when an SDO download was aborted. (segmented transfer)

2.6.13 ESS_OD_OBJECT_INFOS

Struct member	Description
name	Object name.
dataType	Object data type, see *COE_DATATYPE*. (Usually: With a *COE_CODE* of *COE_CODE_VARIABLE*/*COE_CODE_ARRAY* the same data type as for entry/entries is used. With *COE_CODE_RECORD* *COE_DATATYPE_INVALID* is used.)
code	Object kind, see *COE_CODE*.

2.6.14 ESS_OD_ENTRY_INFOS

Struct member	Description
name	Entry name.
minValue	Optional pointer to *uint8_t* array that contains the minimum value for that entry. Data must remain there, must be stored as little-endian and must have same length as the entry.
maxValue	Optional pointer to *uint8_t* array that contains the maximum value for that entry. Data must remain there, must be stored as little-endian and must have same length as the entry.
defaultValue	Optional pointer to *uint8_t* array that contains the default value for that entry. Data must remain there, must be stored as little-endian and must have same length as the entry.
unitType	Unit type value. 0 when this info shall not be included, see [ETG.1004] for unit details.
dataType	Entry's data type, see *COE_DATATYPE*.

2.6.15 ESS_CONFIG_FLAGS

#define	Description
ESS_CONFIG_FLAGS_NONE	No special configuration flags.
ESS_CONFIG_FLAGS_USE_ISR	HAL shall use ISR instead of polling ESC registers. (Based on used HAL this might be required or forbidden – refer to used HAL's "Readme", etc.)
ESS_CONFIG_FLAGS_APPSUPP_BOOTSTRAP	Application supports bootstrap mode. Note: No special settings are required for Bootstrap mode, i.e. standard mailbox SM settings are used. (Mailbox settings for Bootstrap mode are stored in the slave's .xml and EEPROM data)

2.6.16 ESS_EVENT

This is only used internally for certain HALs. With *CFG_HAL_NEEDS_EVENTS* the slave stack triggers these events and it's up to the HAL to use/handle them.

2.6.17 ESS_DC_EVENT

#define	Description
ESS_DC_EVENT_SYNC0	Bit 2 in ESC register 0x0220 was set.
ESS_DC_EVENT_SYNC1	Bit 3 in ESC register 0x0220 was set.
ESS_DC_EVENT_LATCH	Bit 1 in ESC register 0x0220 was set.

2.6.18 ESS_MBX_PACKET

See [ETG.1000.6] for more mailbox transfer details.

Struct member	Description
length	Mailbox Header: Length of packet data.
address	Mailbox Header: Station address.
channelAndPrio	Mailbox Header: Channel and priority.
typeAndCounter	Mailbox Header: Type and counter.
data	Mailbox data, *length* bytes used.

2.6.19 ESS_CBDATA_CYCLIC

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
essTime	Set by HAL before calling the callback. Always in milliseconds, but accuracy/jitter depends on HAL/hardware.

2.6.20 ESS_CBDATA_COE_EVENT

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
abortCode	Allows the application to abort the access. Only supported for *ESS_OD_ENTRY_CALLBACK_STARTING_DOWNLOAD/UPLOAD* and for objects/entries that were created by *essODObjectAdd()* and *essODEntryAdd()*.
objIdx	Object index.
cbType	Callback type, see *ESS_OD_ENTRY_CALLBACK*.
entryIdx	Entry index.

2.6.21 ESS_COE_EMERGENCY

Struct member	Description
errCode	Value for CoE emergency error code. An overview can be found in [ETG.1000.6].
errReg	Value for CoE emergency error register.
data	Value for CoE emergency error data.

2.6.22 ESS_PDO_ENTRY

Struct member	Description
objIdx	Index of object to be mapped.
subIdx	Sub index of object to be mapped.
len	Length of mapped entry in bit.

The API defines three macros to make PDO mapping definition more readable (see 2.3.3):

ESS_MAP_ENTRY(objIdx, subIdx, len):

Define mapping of an existing object with less or equal 31 bytes (248 bit) or define the mapping of the first part (240 bits) of an object with more than 31 bytes.

ESS_MAP_EXTEND(size):

Continue the definition of a mapping for an object with more than 31 bytes in chunks of 240 bit apart from the last chunk.

ESS_MAP_DUMMY(size):

Define a dummy mapping entry (for data alignment) with *size* bits.

2.6.23 ESS_CBDATA_STATE_REQUEST

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
transition	The state transition (*ESC_TRANSITION*) that is requested.
statusCode	Error code that was detected by stack. When *0* state is changed after finishing this callback. When application itself detects an error that prevents the state change it sets *statusCode* accordingly (When stack already set it, the application must not change it – no application defined state change possible then!) See *REG_VAL_ALSTATUSCODE* for constants to use.
newState	The new state that was requested. The application can change this value when it detects an error. (And when stack did not find one at first. And also only to a lower state / according to the EtherCAT rules!)
errAck	*Err Ind Ack* value from AL Control register (0x0120) that was written by the master.
tryAgain	Application can set this to *ESS_TRUE* when it is not yet able to complete the callback – the stack will then try again until it is set to *ESS_FALSE*.
devIdRequest	Set to *ESS_TRUE* by the stack if the master has requested to store the device ID in the AL Status Code Register (0x0134) according to the 'Explicit Device ID' mechanism defined in ETG.1020.
devId	The application stores the current configured device ID in this member variable if *devIdRequest* is *ESS_TRUE*.

2.6.24 ESS_CBDATA_SM_EVENT

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
sm	Bit mask which SM triggered the callback (Bit x: SMx with x 0..15). If more than one SM is configured for input/output process data more than one bit may be set. If passed to the *cbOutputsUpdated()* callback (see 2.5) the bit ESS_SM_EVENT_FLAG_SAFE_OUTPUTS is set to indicate that the slave is in the SAFEOP state and the application should set the application specific safe-state values instead of the values received by the EtherCAT master.

2.6.25 ESS_CBDATA_INOUTPUTS_ACTIVATE

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
inputs	When *ESS_FALSE* then callback was triggered for outputs else for inputs.
started	When *ESS_FALSE* outputs/inputs are stopped else started. An application which supports input process data as

2.6.26 ESS_CBDATA_SM

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
smPhysAddr	Physical address of SM this callback is for.
smLen	Length of that SM in bytes.
smIdx	SM index.
smContrByte	SM control byte (ESC register 0x0804, etc.)
smType	SM type, e.g. *SM_TYPE_OUTPUTS*.
smActivate	Value of SM Activate register (ESC register 0x0806, etc.)

2.6.27 ESS_CBDATA_COE_READWRITE

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
destSrc	When *isRead* is *ESS_TRUE*: Destination where to copy entry data to, else source where to read entry data from. (Keep in mind: EtherCAT uses little endian)
offset	Current byte offset in entry's data.
abortCode	Application may set this when it wants to abort the SDO transfer. See *COE_ABORTCODE* for possible values.
len	When *isRead* is *ESS_TRUE*: Length of data that shall be written to *destSrc*, else length of data available at *destSrc*.
objIdx	Index of object the entry belongs to.
entryIdx	Entry index.
isRead	Whether it's a read (SDO upload) or write access (SDO download) to the entry.
isLastPart	Whether it's the last part of the SDO transfer or not.

2.6.28 ESS_CBDATA_FOE_OPEN

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
fileName	Name of file that is to be opened, zero terminated.
appHandle	Custom value that can be set by the application.
password	Password for the file.
errCode	Application has to write this to tell the stack whether the access is allowed or has to be aborted, see *FOE_ERRORCODE* for possible values.
errorText	Application sets this in conjunction with *errCode*.
fileNameLen	Length of file name. (as *strlen()* would count it)
isUpload	Whether this is a FoE upload or download request.
maxDataLen	Net size of the mailbox. The end of an FoE upload/download transfer is indicated by / has to be indicated to the master with a data segment which size is smaller than this mailbox size (or even 0) before the FoE cloase handler is called. The application can use this parameter to detect this situation. This parameter is also provided in ESS_CBDATA_FOE_DATA.

2.6.29 ESS_CBDATA_FOE_CLOSE

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
appHandle	Custom application value that was set during the FoE open request.
isUpload	Whether this is for a FoE upload or download request.
unexpected	*ESS_TRUE* when the FoE transfer was aborted/timed out.

2.6.30 ESS_CBDATA_FOE_DATA

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
appHandle	Custom application value that was set during the FoE open request.
errorText	Application may return an (optional) error or busy situation related string in conjunction with an *errCode* or returns NULL to provide no further textual information to the EtherCAT master.
destSrc	When *isUpload* is *ESS_TRUE*: Destination where to copy entry data to, else source where to read entry data from.
offset	Current byte offset in file data.
errCode	Application has to return this to tell the stack whether the data transfer shall continue (by returning *FOE_ERRCODE_NONE)* or has to be terminated, see *FOE_ERRORCODE* for possible values. The EtherCAT FoE service also allows to indicate a '***busy'*** situation at any time during the data phase to the EtherCAT master with the (optional) information how many percent (0..100) of the expected busy phase has already elapsed if the data is either not yet available or the data can not yet be stored. The application has to use the macro FOE_ERRCODE_BUSY(arg) with the argument set to a value between 0..100. As a result the handler will be called again with the same values for *offset* and *len* until the request is completed with a *FOE_ERRORCODE* or the *ESS_CB_FOE_CLOSE* handler is called as the ErtherCAT master has terminated the transfer.
len	When *isUpload* is *ESS_TRUE*: Length of data that shall be written to *destSrc* (writing less tells the stack upload is completed by that chunk), else length of data available at *destSrc*.
isUpload	Whether this is a FoE upload or download request.
maxDataLen	Net size of the mailbox. The end of an FoE upload/download transfer is indicated by / has to be indicated to the master with a data segment which size is smaller than this mailbox size (or even 0) before the FoE cloase handler is called. The application can use this parameter to detect this situation. This parameter is also provided in ESS_CBDATA_FOE_OPEN.

2.6.31 ESS_CBDATA_EOE_SETIPPARAM

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
macAddress	Pointer to MAC address. (or *NULL* when this was not included in the request)
ipAddress	Pointer to IP address. (or *NULL* when this was not included in the request)
subnetMask	Pointer to subnet mask. (or *NULL* when this was not included in the request)
defaultGateway	Pointer to default gateway IP address. (or *NULL* when this was not included in the request)
dnsServerIP	Pointer to DNS server IP address. (or *NULL* when this was not included in the request)
dnsName	Pointer to DNS name. (or *NULL* when this was not included in the request)
result	Set by application to determine the result of the EoE request, see *EOE_RESULTCODE* for possible values.

2.6.32 ESS_CBDATA_EOE_SETADDRFILTER

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
macAddresses	Pointer to MAC addresses. Count is determined by *macAddrFilterCount* member.
macAddrFilter	Pointer to MAC address filters. Count is determined by *macAddrFilterCount* member.
result	Set by application to determine the result of the EoE request, see *EOE_RESULTCODE* for possible values.
macAddrCount	Number of MAC addresses at *macAddresses*.
macAddrFilterCount	Number of MAC address filters at *macAddrFilter*.
filterBroadcasts	Whether to filter broadcast messages or not.

2.6.33 ESS_CBDATA_EOE_FRAME

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
data	Pointer to the Ethernet frame data.
reserved1	Currently unused
dataLen	Number of bytes (complete Ethernet frame) at *data*.
frameNo	Frame number, taken from EoE request header.
reserved2	Currently unused

2.6.34 ESS_CBDATA_AOE

2.6.35 ESS_CBDATA_SOE

2.6.36 ESS_CBDATA_VOE

All three structs are identical.

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
mb	Pointer to mailbox packet that was received, see *ESS_MBX_PACKET*. To send a reply replace its *data* contents as desired and set result to *MBX_ERR_SUCCESS*.
mbMaxLen	Available space at *mb->data* (the reply length is given by *mb->length*)
result	Mailbox result. If *MBX_ERR_SUCCESS*, then *mb* is sent as reply.
fragment	Set this to *ESS_TRUE* if the reply is fragmented, i.e. another callback is needed. If this is initially set it indicates the callback was triggered by former *fragment=ESS_TRUE* instead of an actual packet (*mb* members are undefined then).

2.6.37 ESS_CBDATA_EEPROM_EMULATION

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
destSrc	Pointer to store data read from emulated EEPROM (*isRead*) or pointer to data to write to emulated EEPROM.
offset	EEPROM offset in byte.
len	Data length in byte.
result	Set to 0 to signal successful access.
isRead	Whether it's a read (application stores read data at *destSrc*) or write access (application reads data to write from *destSrc*).

2.6.38 ESS_CBDATA_DC

Struct member	Description
hDev	Handle to device for which the callback was set. (Needed only when same callback function is used for multiple devices)
evtType	See ESS_DC_EVENT.

2.6.39 ESS_CB_STATE_REQUEST

Function definition:

Name	Result	Parameters
ESS_CB_STATE_REQUEST	*void*	*(ESS_CBDATA_STATE_REQUEST*)*

2.6.40 ESS_CB_SYNCMANAGER

Function definition:

Name	Result	Parameters
ESS_CB_SYNCMANAGER	*ESS_BOOL*	*(ESS_CBDATA_SM*)*

Result: *ESS_TRUE* when application handled that SM interrupt and stack should not process it.

2.6.41 ESS_CB_OUTPUTS_UPDATED

Function definition:

Name	Result	Parameters
ESS_CB_OUTPUTS_UPDATED	*void*	*(ESS_CBDATA_SM_EVENT*)*

2.6.42 ESS_CB_INPUTS_UPDATED

Function definition:

Name	Result	Parameters
ESS_CB_INPUTS_UPDATED	*void*	*(ESS_CBDATA_SM_EVENT*)*

2.6.43 ESS_CB_COE_EVENT

Function definition:

Name	Result	Parameters
ESS_CB_COE_EVENT	*void*	*(ESS_CBDATA_COE_EVENT*)*

2.6.44 ESS_CB_COE_READWRITE

Function definition:

Name	Result	Parameters
ESS_CB_COE_READWRITE	*void*	*(ESS_CBDATA_COE_READWRITE*)*

2.6.45 ESS_CB_FOE_OPEN

Function definition:

Name	Result	Parameters
ESS_CB_FOE_OPEN	*void*	*(ESS_CBDATA_FOE_OPEN*)*

2.6.46 ESS_CB_FOE_CLOSE

Function definition:

Name	Result	Parameters
ESS_CB_FOE_CLOSE	*void*	*(ESS_CBDATA_FOE_CLOSE*)*

2.6.47 ESS_CB_FOE_DATA

Function definition:

Name	Result	Parameters
ESS_CB_FOE_DATA	*void*	*(ESS_CBDATA_FOE_DATA*)*

2.6.48 ESS_CB_INOUTPUTS_ACTIVATE

Function definition:

Name	Result	Parameters
ESS_CB_INOUTPUTS_ACTIVATE	*void*	*(ESS_CBDATA_INOUTPUTS_ACTIVATE*)*

2.6.49 ESS_CB_CYCLIC

Function definition:

Name	Result	Parameters
ESS_CB_CYCLIC	*void*	*(ESS_CBDATA_CYCLIC*)*

2.6.50 ESS_CB_EOE_SETIPPARAM

Function definition:

Name	Result	Parameters
ESS_CB_EOE_SETIPPARAM	*void*	*(ESS_CBDATA_EOE_SETIPPARAM*)*

2.6.51 ESS_CB_EOE_SETADDRFILTER

Function definition:

Name	Result	Parameters
ESS_CB_EOE_SETADDRFILTER	*void*	*(ESS_CBDATA_EOE_SETADDRFILTER*)*

2.6.52 ESS_CB_EOE_FRAME

Function definition:

Name	Result	Parameters
ESS_CB_EOE_FRAME	*void*	*(ESS_CBDATA_EOE_FRAME*)*

2.6.53 ESS_CB_AOE

Function definition:

Name	Result	Parameters
ESS_CB_AOE	*void*	*(ESS_CBDATA_AOE*)*

2.6.54 ESS_CB_VOE

Function definition:

Name	Result	Parameters
ESS_CB_VOE	*void*	*(ESS_CBDATA_VOE*)*

2.6.55 ESS_CB_SOE

Function definition:

Name	Result	Parameters
ESS_CB_SOE	*void*	*(ESS_CBDATA_SOE*)*

2.6.56 ESS_CB_EEPROM_EMULATION

Function definition:

Name	Result	Parameters
ESS_CB_EEPROM_EMULATION	*void*	*(ESS_CBDATA_EEPROM_EMULATION*)*

2.6.57 ESS_CB_DC

Function definition:

Name	Result	Parameters
ESS_CB_DC	*void*	*(ESS_CBDATA_DC*)*

2.6.58 ESS_STATISTICS

Struct member	Description
smInterrupts	Counter for SM events. SM0 events are counted in *smInterrupts[0]*, SM1 events in *smInterrupts[1]* and so on. Incremented whenever the stack sees the according bit in ESC Reg. 0x0220 set, i.e. not necessarily by PDI interrupt
totalCalls	Counts how often the stack's main loop was called
cyclicCalls	Counts how often the stack's main loop was called by cyclic event handler (which equals the number of calls to the cbCyclic callback)
eoeSent	Counts EoE frames sent. (Incremented when last fragment was written to own mailbox)
eoeReceived	Counts EoE frames received. (Incremented when last fragment was received – just before application callback is triggered)

2.6.59 ESS_SM_CONFIGURATION

Values must match parameter of the related ESI file – stack verifies them against values received by master and denies state change, etc. when they don't match.

Struct member	Description
minSize	Minimum SM length. See remark to mailbox configuration in chapter 2.2.5.
defSize	Default SM length (Not used)
maxSize	Maximum SM length (*0* to disable verification)
startAddr	SM Start address (*0* to disable verification)
contrByte	SM control byte, contains SM type, etc. (see ESC documentation for its register 0x0804)

2.6.60 ESS_CALLBACKS

All *ESS_CALLBACKS* members are described in section 2.5 (page 53).

2.6.61 ESS_CONFIGURATION

Based on used HAL certain requirements might apply, e.g. a minimum value for *timerInterval* or certain *flags* set – refer to used HAL's "ReadMe", etc.

Struct member	Description
essABIVersion	Use *ESS_ABI_VERSION* from *essConfig.h*
flags	Configuration flags, see *ESS_OD_OBJECT_INFOS*
timerInterval	Interval for cyclic callback, in microseconds
tag	User defined pointer, retrieved by *essGetTag()*
cb	Callbacks, see 2.5
smConfigs	Pointer to SM configuration array, see *ESS_SM_CONFIGURATION*
smConfigCount	Number of entries in *smConfigs* for the standard configuration.
smConfigCountBootstrap	Number of entries in *smConfigs* for the bootstrap configuration.
reserved	All reserved members must be set to *0*
stats	Statistics for certain stack events, see *ESS_STATISTICS* (Used with *CFG_ESS_SERVE_STATISTICS*)

2.6.62 ESC_STATE

#define	Description
ESC_STATE_INIT	EtherCAT state "Init"
ESC_STATE_PREOP	EtherCAT state "Pre-Operational"
ESC_STATE_BOOT	EtherCAT state "Bootstrap"
ESC_STATE_SAFEOP	EtherCAT state "Safe-Operational"
ESC_STATE_OP	EtherCAT state "Operational"

2.6.63 ESC_TRANSITION

#define	Description
ESC_TRANSITION_BI	Transition from EtherCAT state “Bootstrap” to “Init”
ESC_TRANSITION_IB	Transition from EtherCAT state “Init” to “Bootstrap”
ESC_TRANSITION_BB	Transition from EtherCAT state “Bootstrap” to “Bootstrap”
ESC_TRANSITION_II	Transition from EtherCAT state “Init” to “Init”
ESC_TRANSITION_IP	Transition from EtherCAT state “Init” to “Pre-Operational”
ESC_TRANSITION_OI	Transition from EtherCAT state “Operational” to “Init”
ESC_TRANSITION_OP	Transition from EtherCAT state “Operational” to “Pre-Operational”
ESC_TRANSITION_OS	Transition from EtherCAT state “Operational” to “Safe-Operational”
ESC_TRANSITION_OO	Transition from EtherCAT state “Operational” to “Operational”
ESC_TRANSITION_PI	Transition from EtherCAT state “Pre-Operational” to “Init”
ESC_TRANSITION_PP	Transition from EtherCAT state “Pre-Operational” to “Pre-Operational”
ESC_TRANSITION_PS	Transition from EtherCAT state “Pre-Operational” to “Safe-Operational”
ESC_TRANSITION_SI	Transition from EtherCAT state “Safe-Operational” to “Init”
ESC_TRANSITION_SO	Transition from EtherCAT state “Safe-Operational” to “Operational”
ESC_TRANSITION_SP	Transition from EtherCAT state “Safe-Operational” to “Pre-Operational”
ESC_TRANSITION_SS	Transition from EtherCAT state “Safe-Operational” to “Safe-Operational”

2.6.64 SM_TYPE

#define	Description
SM_TYPE_INPUTS	SM is of type “Mailbox Inputs”
SM_TYPE_MBXIN	SM is of type “Mailbox In”
SM_TYPE_OUTPUTS	SM is of type “Outputs”
SM_TYPE_MBXOUT	SM is of type “Mailbox Out”

2.6.65 ESS_SM

#define	Description
ESS_SM_NONE	SM not configured/existing
ESS_SM_0	SM0
⋮	
ESS_SM_31	SM31

2.6.66 REG_VAL_ALSTATUSCODE

These are constants for the AL Status Codes which are stored in the AL Status Code Register (0x0134:0x0135) to indicate the reason for an error situation to the EtherCAT master. Most of the status codes are defined in [ETG.1000.6] with some additional codes defined in [ETG.1020].

The numerical range from 0x8000..0xFFFF is reserved for vendor specific AL Status Codes. To define such a code use the macro `REG_VAL_ALSTATUSCODE_VENDOR`. The esd EtherCAT slave stack defines some AL Status Codes starting with the numerical value 0xB000 and reserves the values up to 0xB0FF for future use. Please make sure that your application/vendor specific AL Status Codes do not overlap.

#define REG_VAL_ALSTATUSCODE_	Description
NOERROR	Value: 0x0000, see “AL Status Codes” in [ETG.1020].
UNSPECIFIEDERROR	Value: 0x0001, see “AL Status Codes” in [ETG.1020].
NOMEMORY	Value: 0x0002, see “AL Status Codes” in [ETG.1020].
INVALIDDEVICESETUP	Value: 0x0003, see “AL Status Codes” in [ETG.1020].
FWINCOMPATIBLE	Value: 0x0006, see “AL Status Codes” in [ETG.1020].
FWUPDATEERROR	Value: 0x0007, see “AL Status Codes” in [ETG.1020].
LICENSEERROR	Value: 0x000E, see “AL Status Codes” in [ETG.1020].
INVALIDALCONTROL	Value: 0x0011, see “AL Status Codes” in [ETG.1020].
UNKNOWNALCONTROL	Value: 0x0012, see “AL Status Codes” in [ETG.1020].
BOOTNOTSUPP	Value: 0x0013, see “AL Status Codes” in [ETG.1020].
NOVALIDFIRMWARE	Value: 0x0014, see “AL Status Codes” in [ETG.1020].
INVALIDMBXCFGINBOOT	Value: 0x0015, see “AL Status Codes” in [ETG.1020].
INVALIDMBXCFGINPREOP	Value: 0x0016, see “AL Status Codes” in [ETG.1020].
INVALIDSMCFG	Value: 0x0017, see “AL Status Codes” in [ETG.1020].
NOVALIDINPUTS	Value: 0x0018, see “AL Status Codes” in [ETG.1020].
NOVALIDOUTPUTS	Value: 0x0019, see “AL Status Codes” in [ETG.1020].
SYNCERROR	Value: 0x001a, see “AL Status Codes” in [ETG.1020].
SMWATCHDOG	Value: 0x001b, see “AL Status Codes” in [ETG.1020].
SYNCTYPESNOTCOMPATIBLE	Value: 0x001c, see “AL Status Codes” in [ETG.1020].
INVALIDSMOUTCFG	Value: 0x001d, see “AL Status Codes” in [ETG.1020].
INVALIDSMINCFG	Value: 0x001e, see “AL Status Codes” in [ETG.1020].
INVALIDWDCFG	Value: 0x001f, see “AL Status Codes” in [ETG.1020].
WAITFORCOLDSTART	Value: 0x0020, see “AL Status Codes” in [ETG.1020].
WAITFORINIT	Value: 0x0021, see “AL Status Codes” in [ETG.1020].
WAITFORPREOP	Value: 0x0022, see “AL Status Codes” in [ETG.1020].
WAITFORSAFEOP	Value: 0x0023, see “AL Status Codes” in [ETG.1020].
INVALIDINPUTMAPPING	Value: 0x0024, see “AL Status Codes” in [ETG.1020].
INVALIDOUTPUTMAPPING	Value: 0x0025, see “AL Status Codes” in [ETG.1020].
INCONSISTENTSETTINGS	Value: 0x0026, see “AL Status Codes” in [ETG.1020].
FREERUNNOTSUPPORTED	Value: 0x0027, see “AL Status Codes” in [ETG.1020].
SYNCHRONNOTSUPPORTED	Value: 0x0028, see “AL Status Codes” in [ETG.1020].

#define REG_VAL_ALSTATUSCODE_	Description
FREERUNNEEDS3BUFFERMODE	Value: 0x0029, see “AL Status Codes” in [ETG.1020].
BACKGROUNDWATCHDOG	Value: 0x002a, see “AL Status Codes” in [ETG.1020].
NOVALIDINPUTSANDOUTPUTS	Value: 0x002b, see “AL Status Codes” in [ETG.1020].
FATALSYNCERROR	Value: 0x002c, see “AL Status Codes” in [ETG.1020].
NOSYNCERROR	Value: 0x002d, see “AL Status Codes” in [ETG.1020].
CYCLETIMETOOSMALL	Value: 0x002e, see “AL Status Codes” in [ETG.1020].
DCINVALIDSYNCCFG	Value: 0x0030, see “AL Status Codes” in [ETG.1020].
DCINVALIDLATCHCFG	Value: 0x0031, see “AL Status Codes” in [ETG.1020].
DCPLLSYNCERROR	Value: 0x0032, see “AL Status Codes” in [ETG.1020].
DCSYNCIOERROR	Value: 0x0033, see “AL Status Codes” in [ETG.1020].
DCSYNCMISSEDERROR	Value: 0x0034, see “AL Status Codes” in [ETG.1020].
DCINVALIDSYNCCYCLETIME	Value: 0x0035, see “AL Status Codes” in [ETG.1020].
DCSYNC0CYCLETIME	Value: 0x0036, see “AL Status Codes” in [ETG.1020].
DCSYNC1CYCLETIME	Value: 0x0037, see “AL Status Codes” in [ETG.1020].
MBXAOE	Value: 0x0041, see “AL Status Codes” in [ETG.1020].
MBXEOE	Value: 0x0042, see “AL Status Codes” in [ETG.1020].
MBXCOE	Value: 0x0043, see “AL Status Codes” in [ETG.1020].
MBXFOE	Value: 0x0044, see “AL Status Codes” in [ETG.1020].
MBXSOE	Value: 0x0045, see “AL Status Codes” in [ETG.1020].
MBXVOE	Value: 0x004f, see “AL Status Codes” in [ETG.1020].
EEPROMNOACCESS	Value: 0x0050, see “AL Status Codes” in [ETG.1020].
EEPROMERROR	Value: 0x0051, see “AL Status Codes” in [ETG.1020].
EXTHWNOTREADY	Value: 0x0052, see “AL Status Codes” in [ETG.1020].
SLVRESTARTEDLOCALLY	Value: 0x0060, see “AL Status Codes” in [ETG.1020].
DEVIDUPDATED	Value: 0x0061, see “AL Status Codes” in [ETG.1020].
MODULEIDENTMISMATCH	Value: 0x0052, see “AL Status Codes” in [ETG.1020].
APPCONTRAVAILABLE	Value: 0x00f0, see “AL Status Codes” in [ETG.1020].
esd specific	
ESD_MAP_ASSIGNMISSING	Value 0xB001: Error while PDO mapping: An assignment object (e.g. 0x1c12/0x1c13) is missing.
ESD_MAP_ASSIGNOBJERR	Value: 0xB002: Error while PDO mapping: An assignment object (e.g. 0x1c12/0x1c13) was not created by *essODUpdatePDOAssignment()*.
ESD_MAP_PDOMISSING	Value: 0xB003: Error while PDO mapping: PDO object (e.g. 0x1600/0x1a00) mentioned in an assignment object does not exist.
ESD_MAP_PDOOBJERR	Value: 0xB004: Error while PDO mapping: PDO object (e.g. 0x1600/0x1a00) mentioned in an assignment object was not created with *essODUpdatePDOConfiguration()*.
ESD_MAP_MAPPEDOBJMISSING	Value: 0xB005: Error while PDO mapping: Object mentioned in a PDO object does not exist.
ESD_MAP_MAPPEDOBJERR	Value: 0xB006: Error while PDO mapping: Object mentioned in a PDO object can not be mapped.
ESD_MAP_MAPPEDENTRYMISSING	Value: 0xB007: Error while PDO mapping: Entry mentioned in a

#define REG_VAL_ALSTATUSCODE_	Description
	PDO object was not found.
ESD_MAP_MAPPEDENTRYERR	Value: 0xB008: Error while PDO mapping: Entry mentioned in a PDO object can not be mapped. (e.g. because it was created with datapointer = NULL)
ESD_CAN_INVALIDHANDLE	Value: 0xB010: An Ntcan handle was invalid.
ESD_OTHERSIDEINVALSTATE	Value: 0xB011: Other side of Bridge/Gateway device is in invalid state
ESD_FLASHERROR	Value: 0xB012: Read or write access to flash memory failed.

2.6.67 COE_CODE

See “Object Code” in ETG documents for more details.

#define	Description
COE_CODE_VARIABLE	CoE object is a variable.
COE_CODE_ARRAY	CoE object may consist of multiple entries of same type. Entry0 for max. used sub index.
COE_CODE_RECORD	CoE objects may consist of multiple entries of different type. Entry0 for max. used sub index.

2.6.68 COE_ACCESS

Used as flags.

#define	Description
COE_ACCESS_R_PREOP	Entry is readable in PreOp.
COE_ACCESS_R_SAFEOP	Entry is readable in SafeOp.
COE_ACCESS_R_OP	Entry is readable in Op.
COE_ACCESS_R	Entry is readable in PreOp, SafeOp and Op.
COE_ACCESS_W_PREOP	Entry is writeable in PreOp.
COE_ACCESS_W_SAFEOP	Entry is writeable in SafeOp.
COE_ACCESS_W_OP	Entry is writeable in Op.
COE_ACCESS_W	Entry is writeable in PreOp, SafeOp and Op.
COE_ACCESS_RW	Entry is readable and writeable in PreOp, SafeOp and Op.
COE_ACCESS_RXMAPPABLE	Entry is mappable as Rx object.
COE_ACCESS_TXMAPPABLE	Entry is mappable as Tx object.
COE_ACCESS_ISBACKUPOBJECT	Entry can be used for backup. (Might be used for device replacement, see [ETG.1020])
COE_ACCESS_ISSETTINGSOBJECT	Entry can be used for settings. (Entry might be downloaded by the master during start up, see [ETG.1020])

2.6.69 COE_DATATYPE

See ETG documents for details.

#define	Description
COE_DATATYPE_INVALID	Value: 0x0000
COE_DATATYPE_BOOL	Value: 0x0001
COE_DATATYPE_SINT	Value: 0x0002
COE_DATATYPE_INT	Value: 0x0003
COE_DATATYPE_DINT	Value: 0x0004
COE_DATATYPE_USINT	Value: 0x0005
COE_DATATYPE_UINT	Value: 0x0006
COE_DATATYPE_UDINT	Value: 0x0007
COE_DATATYPE_REAL	Value: 0x0008
COE_DATATYPE_STRING	Value: 0x0009
COE_DATATYPE_OCTETSTRING	Value: 0x000a
COE_DATATYPE_WSTRING	Value: 0x000b
COE_DATATYPE_TIMEOFDAY	Value: 0x000c
COE_DATATYPE_TIMEDIFF	Value: 0x000d
COE_DATATYPE_DOMAIN	Value: 0x000f
COE_DATATYPE_INT24	Value: 0x0010
COE_DATATYPE_LREAL	Value: 0x0011
COE_DATATYPE_INT40	Value: 0x0012
COE_DATATYPE_INT48	Value: 0x0013
COE_DATATYPE_INT56	Value: 0x0014
COE_DATATYPE_LINT	Value: 0x0015
COE_DATATYPE_UINT24	Value: 0x0016
COE_DATATYPE_UINT40	Value: 0x0018
COE_DATATYPE_UINT48	Value: 0x0019
COE_DATATYPE_UINT56	Value: 0x001a
COE_DATATYPE_ULINT	Value: 0x001b
COE_DATATYPE_PDOCOMMPAR	Value: 0x0020
COE_DATATYPE_PDOMAPPING	Value: 0x0021
COE_DATATYPE_SDOPARAMETER	Value: 0x0022
COE_DATATYPE_IDENTITY	Value: 0x0023
COE_DATATYPE_COMMANDPAR	Value: 0x0025
COE_DATATYPE_SYNCPAR	Value: 0x0029
COE_DATATYPE_BIT1	Value: 0x0030
COE_DATATYPE_BIT2	Value: 0x0031
COE_DATATYPE_BIT3	Value: 0x0032

#define	Description
COE_DATATYPE_BIT4	Value: 0x0033
COE_DATATYPE_BIT5	Value: 0x0034
COE_DATATYPE_BIT6	Value: 0x0035
COE_DATATYPE_BIT7	Value: 0x0036
COE_DATATYPE_BIT8	Value: 0x0037
COE_DATATYPE_ENUMFIRST	Value: 0x0800
COE_DATATYPE_ENUMLAST	Value: 0x0fff

2.6.70 COE_ABORTCODE

See ETG documents for details.

#define	Description
COE_ABORTCODE_NONE	Value: 0x00000000
COE_ABORTCODE_TOGGLE	Value: 0x05030000
COE_ABORTCODE_TIMEOUT	Value: 0x05040000
COE_ABORTCODE_CCS_SCS	Value: 0x05040001
COE_ABORTCODE_MEMORY	Value: 0x05040005
COE_ABORTCODE_ACCESS	Value: 0x06010000
COE_ABORTCODE_WRITEONLY	Value: 0x06010001
COE_ABORTCODE_READONLY	Value: 0x06010002
COE_ABORTCODE_SI0_MUSTBE0_FOR_WRITEACCESS	Value: 0x06010003
COE_ABORTCODE_UNSUPP_TYPE_FOR_COMPL_ACCESS	Value: 0x06010004
COE_ABORTCODE_MAILBOX_TOO_SMALL	Value: 0x06010005
COE_ABORTCODE_INVALID_STATE_FOR_MAPPED_OBJ	Value: 0x06010006
COE_ABORTCODE_INDEX	Value: 0x06020000
COE_ABORTCODE_PDO_MAP	Value: 0x06040041
COE_ABORTCODE_PDO_LEN	Value: 0x06040042
COE_ABORTCODE_P_INCOMP	Value: 0x06040043
COE_ABORTCODE_I_INCOMP	Value: 0x06040047
COE_ABORTCODE_HARDWARE	Value: 0x06060000
COE_ABORTCODE_LEN	Value: 0x06070010
COE_ABORTCODE_LEN_TOO_HIGH	Value: 0x06070012
COE_ABORTCODE_LEN_TOO_LOW	Value: 0x06070013
COE_ABORTCODE_SUBINDEX	Value: 0x06090011
COE_ABORTCODE_DATA	Value: 0x06090030
COE_ABORTCODE_DATA_TOO_HIGH	Value: 0x06090031
COE_ABORTCODE_DATA_TOO_LOW	Value: 0x06090032
COE_ABORTCODE_MODULE_MISMATCH	Value: 0x06090033
COE_ABORTCODE_MINMAX	Value: 0x06090036
COE_ABORTCODE_GENERAL	Value: 0x08000000
COE_ABORTCODE_STORE	Value: 0x08000020
COE_ABORTCODE_STORE_LOCAL	Value: 0x08000021
COE_ABORTCODE_STORE_DEVICE_STATE	Value: 0x08000022
COE_ABORTCODE_DICTIONARY	Value: 0x08000023

2.6.71 FOE_ERRORCODE

See ETG documents for details.

#define	Description
FOE_ERRCODE_NONE	Value: 0x00000000
FOE_ERRCODE_UNDEFINED	Value: 0x00008000
FOE_ERRCODE_NOTFOUND	Value: 0x00008001
FOE_ERRCODE_ACCESSDENIED	Value: 0x00008002
FOE_ERRCODE_DISKFULL	Value: 0x00008003
FOE_ERRCODE_ILLEGAL	Value: 0x00008004
FOE_ERRCODE_WRONGPACKETNO	Value: 0x00008005
FOE_ERRCODE_ALREADYEXISTS	Value: 0x00008006
FOE_ERRCODE_NOUSER	Value: 0x00008007
FOE_ERRCODE_BOOTSTRAPONLY	Value: 0x00008008
FOE_ERRCODE_NOBOOTSTRAP	Value: 0x00008009
FOE_ERRCODE_NORIGHTS	Value: 0x0000800a
FOE_ERRCODE_PROGRAMERROR	Value: 0x0000800b
FOE_ERRCODE_INVALIDCHECKSUM	Value: 0x0000800c
FOE_ERRCODE_INVALIDFIRMWARE	Value: 0x0000800e
FOE_ERRCODE_NOFILE	Value: 0x0000800f
FOE_ERRCODE_UNKNOWNHEADER	Value: 0x00008010
FOE_ERRCODE_FLASHPROBLEM	Value: 0x00008011
FOE_ERRCODE_INCOMPATIBLE	Value: 0x00008012

2.6.72 EOE_RESULTCODE

See ETG documents for details.

#define	Description
EOE_RESULTCODE_SUCCESS	Value: 0x0000
EOE_RESULTCODE_UNSPECIFIEDERR	Value: 0x0001
EOE_RESULTCODE_UNSUPPORTEDFRAME	Value: 0x0002
EOE_RESULTCODE_NOIPSUPPORT	Value: 0x0201
EOE_RESULTCODE_NOFILTERSUPPORT	Value: 0x0401

3. Object version specific

Object versions consist of the stack as binary static or shared library (a *.dll* file for Windows, a *.so* file for Linux and *.a* file for VxWorks 7) and an additional device driver specific for the EtherCAT hardware.

3.1 Build

The *.../build* directory of the stack installation contains sample project files for Microsoft Visual Studio 10 and sample Makefiles tested with the GCC tool chain.

They have to be adapted to your specific environment. Currently the Makefiles can be used e.g. with "*make PLATFORM=WINDOWS*" or "*make PLATFORM=LINUX*" (Where the latter one also uses the *CROSS_COMPILE* environment variable.) and might already work for some platforms.

For VxWorks 7 you have to create an appropriate project with the *Wind River Workbench* and link to this with the static library of the esd EtherCAT slave stack.

The *essConfig.h* file (described in next section) can't be changed with the object versions, but it might show useful hints about the parameters the stack was built with.

4. Source Code Version specific

The Source Code Version usually requires the customization of the HAL to the target system. Fig. 2 gives an overview of the general code flow: the Application can remain unchanged, but the Stack's functions to access the ESC, interrupts etc. have to be adapted.

This section shows how the Stack is build from the Source Code and what parts have to be adapted. (A general usage overview was given in 2.1)

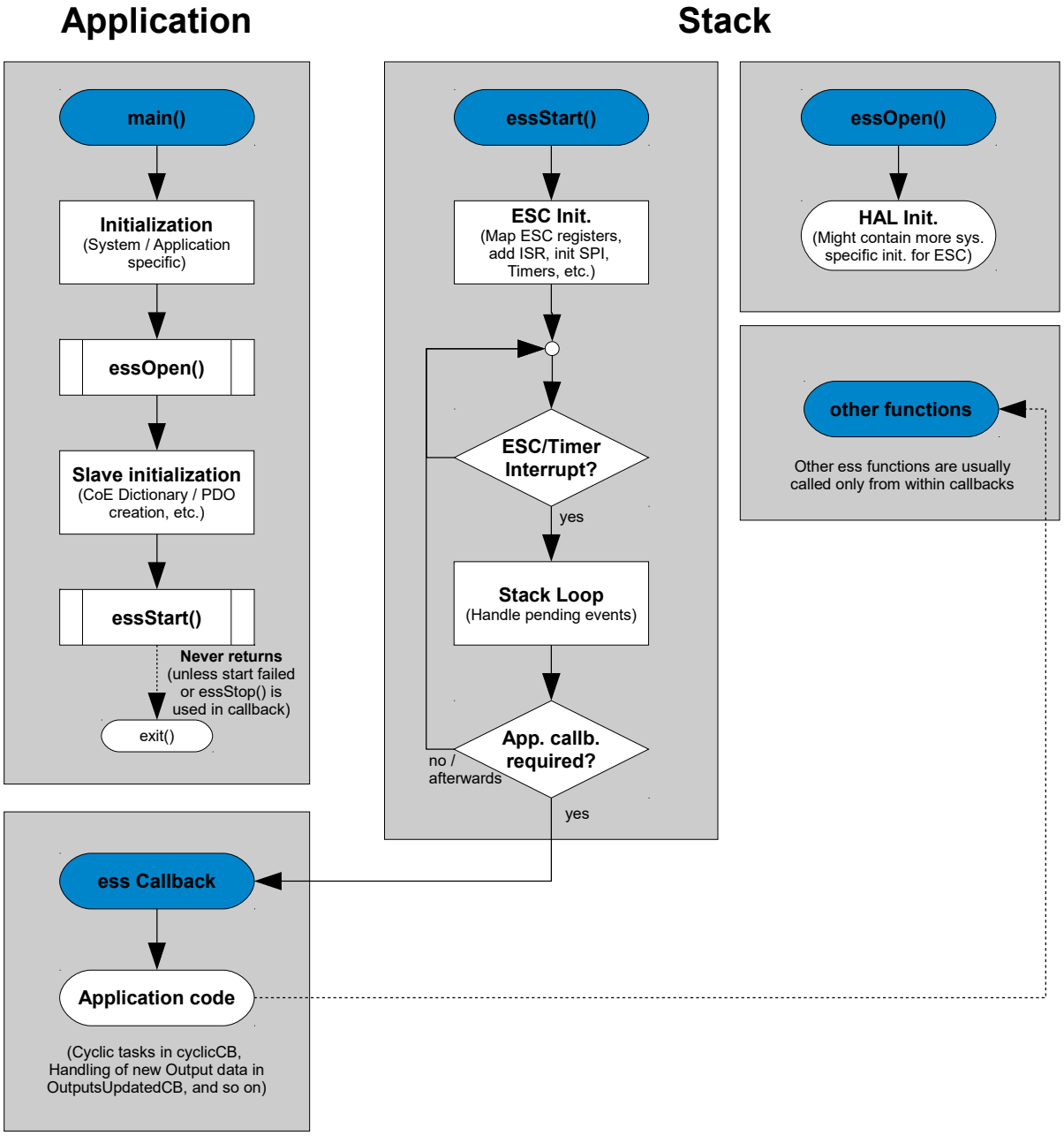

Fig. 2: Stack/Application flow chart

4.1 Build

The stack consists of three source files that have to be compiled/linked:

1. *.../src/ess.c*
2. *.../src/essOD.c*
3. *.../hal/[YourPlatform]/essHAL.c*

   (Hardware/platform specific, e.g. in *.../hal/linux/essHAL.c* for Linux. See section HAL for details to add an implementation for a yet unsupported platform)

Two include paths have to be added: (Fig. 3 gives an overview)

1. *.../include/*
2. *.../hal/*

.../include/essSystem.h has to be adapted:

This file is included by all stack sources as well as your application. It contains the platform specific includes, defines and macros. It is controlled by the compile time define *ESS_PLATFORM_XXX*: which will result in a compile time error in case it is undefined. The following platforms are already supported:

#define	Description
ESS_PLATFORM_WINDOWS	Windows
ESS_PLATFORM_LINUX	Linux
ESS_PLATFORM_AM335X	TI Sitara AM335x @ TI-RTOS (SYSBIOS)
ESS_PLATFORM_QNX	QNX
ESS_PLATFORM_RIN32M3	Renesas R-IN32M3
ESS_PLATFORM_XMC4800	Infineon XMC4800
ESS_PLATFORM_VXWORKS	VxWorks

To support your yet unsupported OS or environment extend this list at the end with an "*#elif defined(YOURPLATFORM)*" etc. and add the required includes, defines and macros there.

.../include/essPrivate.h might need changes, too:

This file contains types and macros that are private to the stack sources (i.e. not for the application). Some macros, such as

```
#define ESS_MEMCOPY(pDst,pSrc,len)   memcpy((pDst),(pSrc),(size_t)(len))
```

have to be verified – replace them if necessary.

4.1.1 Example

Building the *complex.c* sample with the Linux HAL by the gcc: (while in *.../apps/* folder)

```
gcc -o sample complex.c ../src/ess.c ../src/essOD.c ../hal/linux/essHAL.c -I../include -I../hal -DLINUX
```

Typically a Makefile is created to add additional directives, such as the configuration defines (e.g. *-DCFG_ESS_SUPPORT_FOE=1* for the gcc sample to enable FoE)

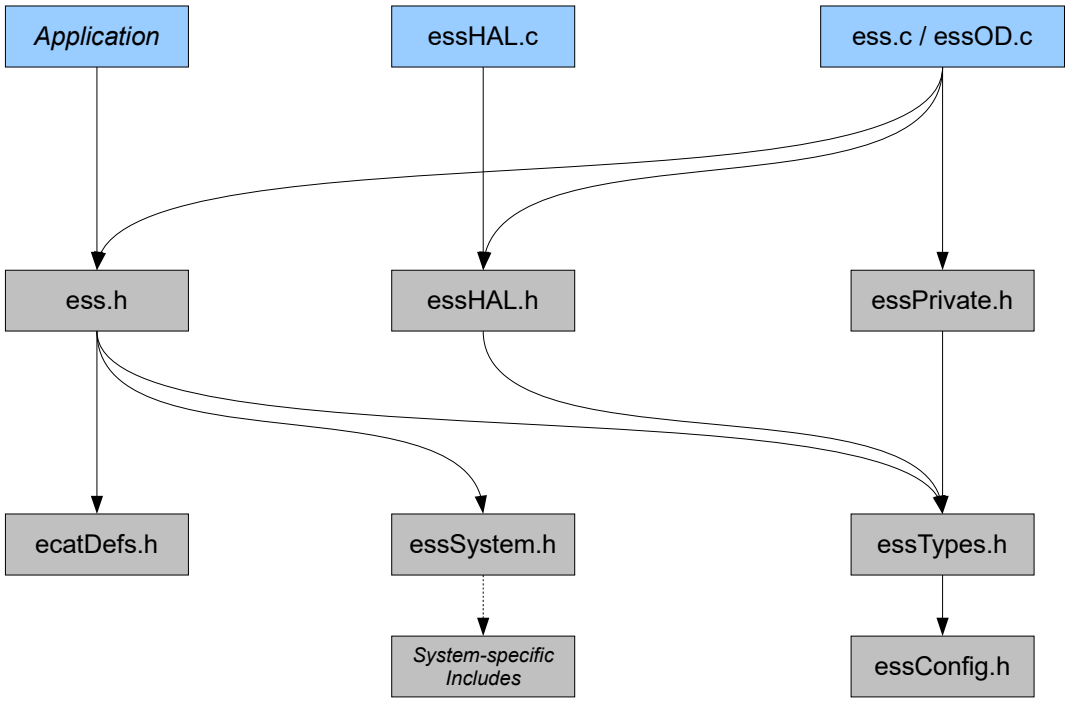

Fig. 3: Source- and include files and their includes.

4.2 essConfig.h

This file is used to configure the stack when building it from source.

#define	Description
CFG_BIG_ENDIAN	Set this to *1* on big endian systems.
CFG_ESC_HAVE_POINTER	Set this to *1* when ESC memory is accessible by a pointer (acquired by *essHALMapESC()* Set to *0* when HAL's *essHALReadESCMem()* / *essHALWriteESCMem()* have to be used instead.
CFG_ESS_COE_ENTRY_LIST_INCREMENT	Number of new items allocated in object's entry list when there's no space for next entry left (each is a pointer)
CFG_ESS_COE_MAPPED_LIST_INCREMENT	For each SM a list of entries mapped to it exists. This is the number of new items allocated when there's no space for the next item left (each is a *ESS_PDO_MAP_INFO* struct)
CFG_ESS_COE_MAX_ITEMS_PER_PDO_OBJECT	Each PDO object (created with *essODUpdatePDOConfiguration()*) will contain an array of *ESS_PDO_ENTRY* with this size.
CFG_ESS_COE_MAX_ITEMS_PER_PDOASSIGN_OBJECT	Each PDO assignment object (created with *essODUpdatePDOAssignment()*) will contain an array of *uint16_t* with this size.
CFG_ESS_COE_OBJECT_LIST_INCREMENT	Number of new items allocated in dictionary's object list when there's no space for next object left (each is a pointer)
CFG_ESS_COE_SDO_SEGMENT_TIMEOUT	Timeout for SDO transfer: when last segment receipt is longer ago than this, transfer is aborted.
CFG_ESS_COE_SUPPORT_COMPLETE_ACCESS	Not implemented, set to *0*.
CFG_ESS_COE_SUPPORT_SDO_INFO_SERVICE	Determines whether the "SDO Information Service" is enabled (This allows the master to read the object dictionary, etc.) Should be disabled only when there are no system resources to handle *ESS_OD_OBJECT_INFOS* and *ESS_OD_ENTRY_INFOS* structs for the objects and entries.
CFG_ESS_COMBINED_RUNERR_LED	If *CFG_ESS_SERVE_ERR_LED* and *CFG_ESS_SERVE_RUN_LED* are set to *1* and both LEDs are combined, then this will disable the RUN LED while the Error LED is in another state than OFF
CFG_ESS_COPY_CALLBACKS	Set this to *0* to save some memory (sizeof *ESS_CALLBACKS*), *1* for minor performance gain (stack saves dereferencing configuration struct pointer because it will copy the callbacks).
CFG_ESS_DC_CALLBACK	Set to *1* when the DC callback (*cbDCEvent* in *ESS_CONFIGURATION/ESS_CALLBACKS*) shall be used.
CFG_ESS_EEPROM_ACK_WAIT	Time to wait before EEPROM command retry, in

#define	Description
	ms. (Happens when ACK missed, max. 10 retries)
CFG_ESS_EEPROM_EMULATION	Determines whether the EEPROM is emulated and the according callback is used.
CFG_ESS_FOE_TRANSFER_TIMEOUT	Timeout for FoE transfers: when last data packet is longer ago than this, transfer is aborted. (in ms)
CFG_ESC_HAVE_READ_FUNC	If set to 1 the stack calls the HAL specific function *essHALReadESCMem()* instead of using the internal code to read ESC memory although *CFG_ESC_HAVE_POINTER* is set to 1.
CFG_ESC_HAVE_WRITE_FUNC	If set to 1 the stack calls the (optional) HAL specific function *essHALWriteESCMem()* instead of using the internal code to read ESC memory although *CFG_ESC_HAVE_POINTER* is set to 1.
CFG_ESS_MAX_DEVICES	Max *devIdx* supported, e.g. in *essOpen()*.
CFG_ESS_MAX_EEPROM_WAIT	Max. time to wait for cleared busy bit during EEPROM accesses, in ms.
CFG_ESS_MAX_MBX_LEN	Max. mailbox length. Max. 1486 byte, min. depends on supported mailbox protocol. (When a very small value is required: 64 should be suitable)
CFG_ESS_MAX_SM_COUNT	Max. sync manager per ESC.
CFG_ESS_MIN_SM_ALLOC	Min size to allocate for SM buffer copy, might be increased to avoid *realloc()* calls.
CFG_ESS_OD_ALLOW_HANDLE_PDO_PARAMS	Determines whether the PDO Parameter functions are enabled. (*essODPDOParamCreate()* etc.)
CFG_ESS_OD_ALLOW_HANDLE_SM_TYPE	Determines whether *ESS_OD_FLAGS_HANDLE_SM_TYPES* is enabled. (See *ESS_OD_FLAGS*)
CFG_ESS_SERVE_CUSTOM_LED	May be set to *0* to completely disable LED code if *CFG_ESS_SERVE_RUN_LED* and *CFG_ESS_SERVE_ERR_LED* is also *0*.
CFG_ESS_SERVE_ERR_LED	Set to *1* to let the stack handle the ERR LED. (Has no effect when HAL has not implemented the LED access)
CFG_ESS_SERVE_RUN_LED	Set to *1* to let the stack handle the RUN LED. (Has no effect when HAL has not implemented the LED access)
CFG_ESS_SERVE_STATISTICS	Set this to *1* when the *stats* member of the *ESS_CONFIGURATION* shall be served. (That member must not be *NULL* then)
CFG_ESS_SM_CALLBACK	Set to *1* to allow the application to handle all SM interrupts. Might be used to “hook” into these interrupts – really replacing that interrupt handling by application would lead to almost writing an own EtherCAT stack.
CFG_ESS_SUPPORT_AOE	Determines whether the AoE callbacks are enabled.

Source Code Version specific

#define	Description
CFG_ESS_SUPPORT_COE	Determines whether the CoE functions are enabled. (When disabled, most of stack functionality is lost – might be useful only for minimalistic slaves with an application that handles even SM buffers, etc. itself)
CFG_ESS_SUPPORT_EOE	Determines whether the EoE functions are enabled.
CFG_ESS_SUPPORT_FOE	Determines whether the FoE functions are enabled.
CFG_ESS_SUPPORT_SOE	Determines whether the SoE callbacks are enabled.
CFG_ESS_SUPPORT_VOE	Determines whether the VoE callbacks are enabled.
CFG_ESS_EXPLICIT_DEVICE_ID	Determines whether the stack supports the device ID request via AL Status Code Register (0x134).
CFG_HAL_EXTERNAL	Set this to *1* when HAL resides in an external library. (Instead of being directly compiled/linked into the stack)
CFG_HAL_NEEDS_EVENTS	Set this to *1* when HAL wants to be informed about certain events, like “Read an SM buffer” etc. (Currently only used for AM335x PRU)
CFG_HAVE_STD_INT_TYPES	Set this to *1* when e.g. *<inttypes.h>* was included previously, else *ess.h* will try to define *int8_t*, etc. itself. The latter is not recommended, as the stack’s definitions are only rudimentary (but *essOpen()* will fail when a type is not of expected size).
CFG_ESS_DISABLE_DEPRECATED	Set this to *1* if functions/macros etc. that exist only for compatibility reasons shall be removed – might save some RAM / code size
CFG_HAL_USE_SYNC0_IRQ	Set to 1 if your HAL should handle a direct SYNC0 indication instead of using SYNC0 indication via PDI interrupt.
CFG_HAL_USE_SYNC1_IRQ	Set to 1 if your HAL should handle a direct SYNC1 indication instead of using SYNC1 indication via PDI interrupt.

4.2.1 Saving RAM

Some defines might be tweaked to decrease RAM usage:

- Lower *CFG_ESS_MAX_DEVICES* to the actual number of devices that shall be supported, usually only *1* slave device is used
- Lower *CFG_ESS_MAX_MBX_LEN* to the actual allowed maximum mailbox size. For many cases a small mailbox (e.g. *128* byte) is suitable
- Set *CFG_ESS_MIN_SM_ALLOC* to the largest of the “Outputs”/”Inputs” SM length.

  Example: Max. 16 byte input data and max. 32 byte output data: set this to *32*
- Set *CFG_ESS_COE_MAX_ITEMS_PER_PDO_OBJECT* to the number of items in your PDO with the most items.

  Example: Two PDOs exist, one has 3 entries, one has 5 entries: set this to *5*
- Set *CFG_ESS_COE_MAX_ITEMS_PER_PDOASSIGN_OBJECT* to the max. number of PDOs assigned to one SM.

  Example: Three PDOs exist, one can only be assigned to “Inputs” SM, the other two can be assigned to “Outputs”: set this to *2*
- Set *CFG_ESS_COE_OBJECT_LIST_INCREMENT* to the total number of objects (not including entries).

  Make sure all objects are counted, especially don’t forget the objects that are created by e.g. essODAddGenericObjects()
- Try to find a reasonable value for *CFG_ESS_COE_ENTRY_LIST_INCREMENT*.

  When an entry is added to an object (by *essODEntryAdd()*) the pointer list within the object is increased by this many items if the entry does not fit any more.

  There’s no general rule for a good value, e.g. with many objects that have only few items a smaller value might save RAM, but it will also increase calls to *realloc()* – which might be even worse
- Set *CFG_ESS_COE_MAPPED_LIST_INCREMENT* to the max. number of entries (not PDOs) that can be mapped to one SM.

  Example: Two input PDOs with 3 entries each exist and they can be mapped at the same time: set this to *6*
- Disable VoE, EoE, etc. when not needed. Set *CFG_ESS_SUPPORT_**x**OE* to *0* then
- When “SDO Information Service” is not needed by your application, disable it by setting *CFG_ESS_COE_SUPPORT_SDO_INFO_SERVICE* to *0*

4.2.2 CFG_ESS_SERVE_ERR_LED

Stack Error LED handling:

LED state	Set when
ESC_LED_STATE_OFF	State changed successfully to Init or Op.
ESC_LED_STATE_BLINK	State change failed.
ESC_LED_STATE_FLASH_1	Application called *essIndicateError()*.

A watchdog time out is not handled by the stack, i.e. the application has to set the error LED to *ESC_LED_STATE_FLASH_2* then manually – by *essSetLEDState()*.

4.2.3 CFG_ESS_SERVE_RUN_LED

Stack RUN LED handling:

LED state	Set when EtherCAT state is
ESC_LED_STATE_OFF	Init
ESC_LED_STATE_FLICKER	BootStrap
ESC_LED_STATE_BLINK	PreOp
ESC_LED_STATE_FLASH_1	SafeOp
ESC_LED_STATE_ON	Op

4.3 HAL

Following functions have to be implemented by the HAL. Use *.../hal/dummy/essHAL.c* as template when creating one from scratch. (*.../hal/linux/essHAL.c* might be used as an example, too)

The *devIdx* parameter starts at *0*, where *0* shall be the first EtherCAT controller found in the system.

reserved parameters will be set to *NULL* by the stack, if this is not the case, HAL shall fail and return *ESS_RESULT_INVALID_ARG*.

4.3.1 essHALOpen()

Called when device is opened. No other HAL function for that *devIdx* will be called before this one.

```
ESS_RESULT essHALOpen(ESS_DEVICE_INDEX devIdx);
```

HAL usually handles global initializations for that dev here.

4.3.2 essHALClose()

Called when device is closed. No other HAL function for that *devIdx* will be called after this one (unless reopened by *essHALOpen()*).

```
ESS_RESULT essHALClose(ESS_DEVICE_INDEX devIdx);
```

HAL usually handles global finalizations for that dev here.

4.3.3 essHALMapESC()

Called only when *CFG_ESC_HAVE_POINTER* is not *0*.

```
ESS_RESULT essHALMapESC(ESS_DEVICE_INDEX devIdx, void** escPointer);
```

Shall map the ESC memory and return a pointer to that memory via the *escPointer* parameter.

4.3.4 essHALUnmapESC()

Called only when *CFG_ESC_HAVE_POINTER* is not *0*.

```
ESS_RESULT essHALUnmapESC(ESS_DEVICE_INDEX devIdx);
```

Might unmap the ESC memory, i.e. stack will not use the pointer acquired by *essHALMapESC()* any more.

4.3.5 essHALGetTime()

```
ESS_TIMESTAMP essHALGetTime(void);
```

Shall return a 32 bit time stamp. In milliseconds, must wrap after *0xffffffff*.

4.3.6 essHALStart()

```
ESS_RESULT essHALStart(ESS_DEVICE_INDEX devIdx, ESS_HAL_CALLBACK* isr,
                       ESS_HAL_CALLBACK* cyclicCallback,
                       uint32_t cyclicInterval, void* cbData);
```

Shall start the stack's main loop, i.e. when successfully started it must return only when *essHALStop()* is called.

Must call *cyclicCallback* every *cyclicInterval* microseconds, and must call *isr* when an interrupt occurs – both with *cbData* as argument.

isr is optional, i.e. it's *NULL* when stack only wants the cyclic callback.

4.3.7 essHALStop()

```
ESS_RESULT essHALStop(ESS_DEVICE_INDEX devIdx);
```

Shall stop the stack's main loop, i.e. stack's call to *essHALStart()* shall return with *ESS_RESULT_SUCCESS*. If called when already stopped it shall return *ESS_RESULT_SUCCESS* too.

4.3.8 essHALReadESCMem()

Called when *CFG_ESC_HAVE_POINTER* is *0*, else *essHALMapESC()* is called during startup to request a pointer to the ESC memory assuming that this memory can be accessed directly. The function is also called if *CFG_ESC_HAVE_READ_FUNC* is 1.

```
void essHALReadESCMem(ESS_DEVICE_INDEX devIdx, uint16_t address,
                      void* dst, uint16_t len);
```

Shall read *len* bytes from ESC memory at *address*, stored at *dst*. No swapping must be performed.

4.3.9 essHALWriteESCMem()

Called when *CFG_ESC_HAVE_POINTER* is *0*, else *essHALMapESC()* is called during startup to request a pointer to the ESC memory assuming that this memory can be accessed directly. The function is also called if *CFG_ESC_HAVE_WRITE_FUNC* is 1.

```
void essHALWriteESCMem(ESS_DEVICE_INDEX devIdx, uint16_t address,
                       const void* src, uint16_t len);
```

Shall write *len* bytes to ESC memory at *address*, taken from *src*. No swapping must be performed.

4.3.10 essHALSetLEDState()

```
ESS_RESULT essHALSetLEDState(ESS_DEVICE_INDEX devIdx, ESC_LED_TYPE ledId,
                             ESS_BOOL ledOn);
```

Shall switch a LED on or off. Shall return *ESS_RESULT_INVALID_ARG* when the given LED is not supported (it's called for all LEDs, whether stack handles them or not).

No LED is mandatory – but LED support should match configuration, e.g. *ESC_LED_TYPE_ERR* should be supported when *CFG_ESS_SERVE_ERR_LED* is set to *1*.

4.3.11 essHALStackEvent()

```
void HAL_CALLTYPE essHALStackEvent(ESS_EVENT_DATA* e)
```

For custom usage by HAL. Stack fires these events when built with *CFG_HAL_NEEDS_EVENTS*.

4.3.12 essHALIoctl()

Used to implement custom/device specific functions – see *essIoctl()* in 2.4.24.

essIoctl() forwards functions it does not handle to the HAL.

```
ESS_RESULT essHALIoctl(ESS_DEVICE_INDEX devIdx, ESS_IOCTL fn,
                       void* data, uint32_t sizeOfData);
```

5. Order information

Type	Properties	Order No.
EtherCAT Slave Stack Source	EtherCAT Slave Stack Source Code Version	P.4520.01
Related Hardware:		
ECS-PCIe/1100	PCI Express board with EtherCAT slave controller ET1100 incl. driver, stack binary and documentation for Windows and Linux	E.1100.02
ECS-PCIe/FPGA	PCI Express board with EtherCAT IP Core incl. driver, stack binary and documentation for Windows, Linux and VxWorks 7 (x86)	E.1106.02 E.1106.04
ECS-PMC/FPGA	PMC board with EtherCAT IP Core incl. driver, stack binary and documentation for Windows, Linux and VxWorks 7 (x86)	E.1104.02
ECS-XMC/FPGA	XMC board with EtherCAT IP Core incl. driver, stack binary and documentation for Windows, Linux and VxWorks 7 (x86)	E.1102.02
EtherCAT CD	Includes driver, documentation and ESI files for all esd EtherCAT products as well as trial versions of the EtherCAT master for different platforms and the configuration/engineering tool EtherCAT Workbench.	E.1101.01

Table 2: Order information

PDF Manuals
Manuals are available in English and usually in German as well. For availability of English manuals see table below.
Please download the manuals as PDF documents from our esd website www.esd.eu for free.

Manuals		
EtherCAT Slave Stack-ME	Manual in English	P.4520.21

Table 3: Available manuals

Printed Manuals
If you need a printout of the manual additionally, please contact our sales team: sales@esd.eu for a quotation. Printed manuals may be ordered for a fee.