SYS TEC
ELECTRONIC

openPOWERLINK: Ethernet POWERLINK Protocol Stack

Software Manual

Edition September 2009

system house for distributed automation

Revision history

Date/ Version	**Section**	**Change**	**Editor**
Nov 2006 / 1.0	all	Creation	D. Krüger
Apr 2008 / 2.0	API	Description for MN added	D. Krüger
Sep 2009 / 3.0	all	LED module added, added description of new PDO error codes, typos fixed, new Edrv interface added which supports Ethernet controllers with auto-response feature	D. Krüger

Contact	**Direct**	**Your local distributor**
Address:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	Please find a list of our distributors under http://www.systec-electronic.com/distributors
Ordering Information:	+49 (3661) 6279-0 info@systec-electronic.com	
Technical Support:	+49 (3661) 6279-0 support@systec-electronic.com	
Fax:	+49 (3661) 62 79 99	
Web Site:	http://www.systec-electronic.com	

3rd Edition JuneSeptember 2009

1 Introduction

1.1 Ethernet POWERLINK

Ethernet POWERLINK is a Real-Time Ethernet field bus system. It is based on the Fast Ethernet Standard IEEE 802.3.

A managing node (MN), which acts as the master in the EPL network, polls the controlled nodes (CN) cyclically. This process takes place in the isochronous phase of the EPL cycle. Immediately after the isochronous phase follows an asynchronous phase for communication which is not time-critical, e.g. TCP/IP communication. The isochronous phase starts with a Start of Cyclic frame on which all nodes are synchronized. This schedule design avoids collisions, which are usually present on Standard Ethernet, and ensures the determinism of the hard real-time communication. It is implemented in the EPL data link layer. The EPL network can be connected via gateways to non real-time networks.

The communication profile of Ethernet POWERLINK is adapted from CANopen. Thus the design principles like process data object (PDO) for exchange of process variables and service data object (SDO) for configuration of remote object dictionaries are reused. All PDOs are exchanged within the isochronous phase similar to the synchronous PDOs of CANopen, because event triggered PDOs would interfere with the hard real-time requirements.

To conform to IEEE 802.3 each EPL device has got a unique MAC address. Additionally each device is assigned a logical node ID. Mostly, this node ID can be configured via node switches on the device. If a particular EPL device implements a TCP/IP stack it gets a private IP address from class C within the network 192.168.100.0 where the host part equals the EPL node ID.

It is assumed that you are familiar with the Ethernet POWERLINK Communication Profile Specification [1].

1.2 Key Features

- Implements Communication profile EPSG DS 1.1.0 [1]
- Data link layer and NMT state machine for Controlled and Managing Nodes
- SDO via UDP and EPL ASnd frames
- Dynamic PDO mapping
- User-configurable object dictionary
- Supports the EPL cycle features async-only CN and multiplexed CN
- Implemented in plain ANSI C
- Modular software structure for simple portability to different target platforms
- Supports target platforms with and without operating system
- Event driven Communication Abstraction Layer
- Provides Generic API for user-application

1.3 Supported object dictionary entries

The EPL stack currently supports the following communication objects of the OD. That means that the EPL stack uses these objects or provides the functionality for these objects, but the application may support additional objects.

Object 1000h: NMT_DeviceType_U32
Object 1001h: ERR_ErrorRegister_U8
Object 1003h: ERR_History_ADOM
Object 1006h: NMT_CycleLen_U32
Object 1008h: NMT_ManufactDevName_VS
Object 1009h: NMT_ManufactHwVers_VS
Object 100Ah: NMT_ManufactSwVers_VS
Object 1018h: NMT_IdentityObject_REC
Object 1030h: NMT_InterfaceGroup_Xh_REC
Object 1C00h: DLL_MNCRCError_REC

Object 1C02h: DLL_MNCycTimeExceed_REC
Object 1C07h: DLL_MNCNLossPResCumCnt_AU32
Object 1C08h: DLL_MNCNLossPResThrCnt_AU32
Object 1C09h: DLL_MNCNLossPResThreshold_AU32
Object 1C0Bh: DLL_CNLossSoC_REC
Object 1C0Fh: DLL_CNCRCError_REC
Object 1C14h: DLL_LossOfFrameTolerance_U32
Object 1F80h: NMT_StartUp_U32
Object 1F81h: NMT_NodeAssignment_AU32
Object 1F82h: NMT_FeatureFlags_U32
Object 1F83h: NMT_EPLVersion_U8
Object 1F84h: NMT_MNDeviceTypeIdList_AU32
Object 1F89h: NMT_BootTime_REC
Object 1F8Ah: NMT_MNCycleTiming_REC
Object 1F8Bh: NMT_MNPReqPayloadLimitList_AU16
Object 1F8Ch: NMT_CurrNMTState_U8
Object 1F8Dh: NMT_PResPayloadLimitList_AU16
Object 1F8Eh: NMT_MNNodeCurrState_AU8
Object 1F8Fh: NMT_MNNodeExpState_AU8
Object 1F92h: NMT_MNCNPResTimeout_AU32
Object 1F93h: NMT_EPLNodeID_REC
Object 1F98h: NMT_CycleTiming_REC
Object 1F99h: NMT_CNBasicEthernetTimeout_U32
Object 1F9Eh: NMT_ResetCmd_U8
Object 1F9Fh: NMT_RequestCmd_REC

2 Application Programming Interface

2.1 Software Structure

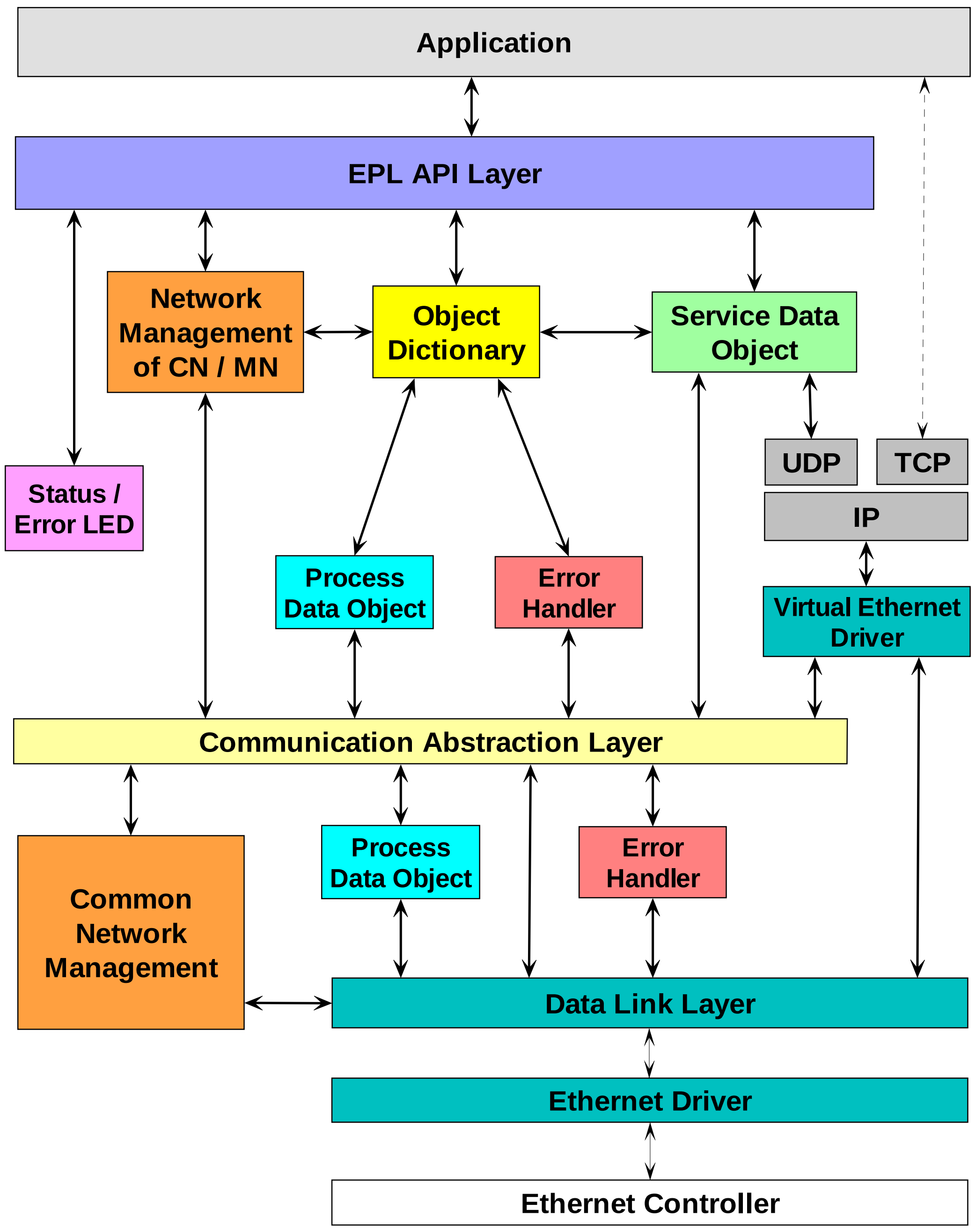

Figure 1: Software structure

The EPL stack is divided into two parts: low-prioritized processes above the Communication Abstraction Layer (abbr. CAL) called EPL user part and high-prioritized processes below the CAL called EPL kernel part. Processes which have to be processed in every EPL cycle have high priority, e.g. Data Link Layer (abbr. DLL), PDO processing and core NMT state machine. All other processes have low priority, e.g. SDO. It is possible to swap out the high-prioritized processes on a separate CPU (e.g. on a SMP machine) to ensure the real-time requirements.

Some modules are divided, i.e. EPL kernel modules have counterparts in EPL user part, which only wrap the communication with the EPL kernel part.

2.1.1 Directory Structure

The source code of the EPL stack is divided in several directories.

Directory	***Description***
Edrv	Ethernet driver implementations
EplStack	EPL protocol stack core components
Example	Example and test projects
Include	Generic header files
Include/kernel	Header files for EPL kernel part
Include/user	Header files for EPL user part
ObjDicts	Sample Object dictionaries
SharedBuff	Shared buffer implementation for CAL and frame queues
Target/ARCH/OS/C	Target dependant files for architecture ARCH, operating system OS and compiler C

Table 1: Directory structure

2.1.2 Module Structure

C File	***Description***
amibe.c	AMI implementation for big endian architectures.
amix86.c	AMI implementation for X86 architecture.
Edrv*.c	Target specific Ethernet driver.
EplApiGeneric.c	Generic implementation of EPL API Layer.
EplApiLinuxKernel.c	Linux kernel module wrapper of EPL API Layer.
EplApiLinuxUser.c	Linux userspace wrapper of EPL API Layer.
EplApiProcessImage.c	Functions for static process image.
EplDllk.c	DLL module in EPL kernel part.
EplDllkCal.c	CAL of DLL module in EPL kernel part.
EplDlluCal.c	CAL of DLL module in EPL user part, e.g. for reception and transmission of EPL ASnd frames.
EplErrorHandlerk.c	Error handler in EPL kernel part. It manages the error counters in OD.
EplEventk.c	Event module in EPL kernel part.
EplEventu.c	Event module in EPL user part.
EplNmtk.c	Common NMT module in EPL kernel part. It manages the NMT state machine.
EplNmtu.c	Wrapper for common NMT module in EPL user part.
EplNmtCnu.c	CN NMT module in EPL user part.
EplNmtMnu.c	MN NMT module in EPL user part.
EplObd.c	OBD module.
EplPdok.c	PDO module in EPL kernel part.
EplPdokCal.c	CAL of PDO module in EPL kernel part.
EplSdoAsndu.c	SDO ASnd protocol layer in EPL user part.
EplSdoAsySequ.c	SDO sequence layer in EPL user part.
EplSdoComu.c	SDO command layer in EPL user part.
EplSdoUdpu.c	SDO UDP protocol layer in EPL user part.
EplTimeruLinuxKernel.c	Timer module implementation for Linux kernel modules in EPL user part. It can be used via define EPL_TIMER_USE_USER in EPL kernel part too.
EplTimeruNull.c	Timer module implementation without any functionality. Only useful for testing purposes.
EplTimeruWin32.c	Timer module implementation for MS Windows in EPL user part. It can be used via define EPL_TIMER_USE_USER in EPL kernel part too.

C File	*Description*
SocketLinxuKernel.c	BSD Socket API for Linux kernel modules.
VirtualEthernetLinux.c	Virtual Ethernet driver implementation for Linux.

Table 2: Module structure

2.1.3 Header files

The EPL stack consists of several header files. But the application only needs to include ***Epl.h***. This header file includes itself all necessary module header files including ***EplCfg.h*** and ***global.h***.

2.1.4 Target dependant modules

The EPL stack was designed under the objective to minimize and encapsulate the target dependant parts. This minimizes and simplifies the porting to new platforms, i.e. hardware or operating systems.

The following modules need to be adapted:
- Ethernet Driver
- Virtual Ethernet Driver
- Communication Abstraction Layer
- Timer modules
- parts of EPL API Layer

2.2 Common data types

2.2.1 tEplNetTime

A frequently used data structure in the EPL stack is tEplNetTime. It represents a timestamp conformant to the data type NETTIME of the EPL specification.

```
typedef struct
{
    DWORD                       m_dwSec;
    DWORD                       m_dwNanoSec;
} tEplNetTime;
```

Member	***Description***
m_dwSec	Seconds.
m_dwNanoSec	Nanoseconds (Range 0 <= bits 0..30 < 10^9; bit 31 represents the sign bit)

Table 3: Members of structure tEplNetTime

2.2.2 tEplKernel

The enumerated type tEplKernel represents the internal error codes. These are defined in the header file EplErrDef.h.

```
typedef enum
{
    kEplSuccessful              = 0x0000,
    kEplInvalidOperation        = 0x0005,
    kEplInvalidNodeId           = 0x0007,
    kEplNoResource              = 0x0008,
    kEplShutdown                = 0x0009,
    kEplReject                  = 0x000A,
    kEplEdrvInitError           = 0x0013,
    kEplEdrvNoFreeBufEntry      = 0x0014,
    kEplEdrvBufNotExisting      = 0x0015,
    kEplEdrvInvalidParam        = 0x001C,
    kEplDllIllegalHdl           = 0x0022,
    kEplDllCbAsyncRegistered    = 0x0023,
    kEplDllAsyncTxBufferEmpty   = 0x0025,
    kEplDllAsyncTxBufferFull    = 0x0026,
    kEplDllNoNodeInfo           = 0x0027,
    kEplDllInvalidParam         = 0x0028,
    kEplDllTxBufNotReady        = 0x002E,
    kEplDllTxFrameInvalid       = 0x002F,
    kEplObdIllegalPart          = 0x0030,
    kEplObdIndexNotExist        = 0x0031,
    kEplObdSubindexNotExist     = 0x0032,
    kEplObdReadViolation        = 0x0033,
    kEplObdWriteViolation       = 0x0034,
    kEplObdAccessViolation      = 0x0035,
    kEplObdUnknownObjectType    = 0x0036,
    kEplObdVarEntryNotExist     = 0x0037,
    kEplObdValueTooLow          = 0x0038,
    kEplObdValueTooHigh         = 0x0039,
    kEplObdValueLengthError     = 0x003A,
    kEplNmtUnknownCommand       = 0x0040,
    kEplNmtInvalidFramePointer  = 0x0041,
    kEplNmtInvalidEvent         = 0x0042,
    kEplNmtInvalidState         = 0x0043,
    kEplNmtInvalidParam         = 0x0044,
    kEplSdoUdpMissCb            = 0x0050,
    kEplSdoUdpNoSocket          = 0x0051,
    kEplSdoUdpSocketError       = 0x0052,
    kEplSdoUdpThreadError       = 0x0053,
    kEplSdoUdpNoFreeHandle      = 0x0054,
    kEplSdoUdpSendError         = 0x0055,
    kEplSdoUdpInvalidHdl        = 0x0056,
    kEplSdoSeqMissCb            = 0x0060,
    kEplSdoSeqNoFreeHandle      = 0x0061,
    kEplSdoSeqInvalidHdl        = 0x0062,
    kEplSdoSeqUnsupportedProt   = 0x0063,
    kEplSdoSeqNoFreeHistory     = 0x0064,
    kEplSdoSeqFrameSizeError    = 0x0065,
    kEplSdoSeqRequestAckNeeded  = 0x0066,
    kEplSdoSeqInvalidFrame      = 0x0067,
    kEplSdoSeqConnectionBusy    = 0x0068,
    kEplSdoSeqInvalidEvent      = 0x0069,
    kEplSdoComUnsupportedProt   = 0x0070,
    kEplSdoComNoFreeHandle      = 0x0071,
    kEplSdoComInvalidHandle     = 0x0073,
    kEplSdoComInvalidSendType   = 0x0074,
    kEplSdoComNotResponsible    = 0x0075,
    kEplSdoComHandleExists      = 0x0076,
    kEplSdoComHandleBusy        = 0x0077,
    kEplSdoComInvalidParam      = 0x0078,
    kEplEventUnknownSink        = 0x0080,
    kEplEventPostError          = 0x0081,
    kEplTimerInvalidHandle      = 0x0090,
    kEplTimerNoTimerCreated     = 0x0091,
    kEplSdoAsndInvalidNodeId    = 0x00A0,
    kEplSdoAsndNoFreeHandle     = 0x00A1,
    kEplSdoAsndInvalidHandle    = 0x00A2,
    kEplPdoNotExist             = 0x00B0,
    kEplPdoLengthExceeded       = 0x00B1,
    kEplPdoGranularityMismatch  = 0x00B2,
    kEplPdoInitError            = 0x00B3,
    kEplPdoConfWhileEnabled     = 0x00B7,
    kEplPdoErrorMapp            = 0x00B8,
    kEplPdoVarNotFound          = 0x00B9,
    kEplPdoSizeMismatch         = 0x00BC,
    kEplPdoTooManyTxPdos        = 0x00BD,
    kEplPdoInvalidObjIndex      = 0x00BE,
    kEplPdoTooManyPdos          = 0x00BF,
    kEplCfgMaConfigError        = 0x00C0,
    kEplCfgMaSdocTimeOutError   = 0x00C1,
    kEplCfgMaInvalidDcf         = 0x00C2,
    kEplCfgMaUnsupportedDcf     = 0x00C3,
    kEplCfgMaConfigWithErrors   = 0x00C4,
    kEplCfgMaNoFreeConfig       = 0x00C5,
    kEplCfgMaNoConfigData       = 0x00C6,
    kEplCfgMaUnsuppDatatypeDcf  = 0x00C7,
    kEplApiTaskDeferred         = 0x0140,
    kEplApiInvalidParam         = 0x0142,

} tEplKernel;
```

Constant	***Description***
kEplSuccessful	Successful termination of the function. No error occurred.
kEplInvalidOperation	The requested operation is not valid in the current situation. Maybe it was request right before and is still running.
kEplInvalidNodeId	Invalid node-ID.
kEplNoResource	No resource available, e.g. out of memory or any other resource from the operating system.
kEplShutdown	Shutdown of the entire stack is requested.
kEplReject	Reject the proceeding operation.
kEplEdrvInitError	Ethernet driver initialization error.
kEplEdrvNoFreeBufEntry	No free buffer entry in Ethernet driver.
kEplEdrvBufNotExisting	Specified buffer does not exist in Ethernet driver.
kEplEdrvInvalidParam	Invalid parameter specified while calling an Ethernet driver function.
kEplDllIllegalHdl	DLL: specified handle is not valid.
kEplDllCbAsyncRegistered	DLL: callback function for asynchronous non-EPL frames was or was not registered before.
kEplDllAsyncTxBufferEmpty	DLL: no Tx frame for transmission available.
kEplDllAsyncTxBufferFull	DLL: Tx buffer is full.
kEplDllNoNodeInfo	DLL: no corresponding node information structure found for the specified node-ID.
kEplDllInvalidParam	DLL: invalid parameters specified on function call.
kEplDllTxBufNotReady	DLL: Tx buffer for PReq is not ready yet.
kEplDllTxFrameInvalid	DLL: Tx frame for PReq is invalid or does not exist.

Constant	***Description***
kEplObdIllegalPart	OBD: illegal OD part referenced.
kEplObdIndexNotExist	OBD: specified object index does not exist.
kEplObdSubindexNotExist	OBD: specified sub index does not exist.
kEplObdReadViolation	OBD: illegal read on a write-only object
kEplObdWriteViolation	OBD: illegal write on a read-only object
kEplObdAccessViolation	OBD: illegal access on an object
kEplObdUnknownObjectType	OBD: unknown object type
kEplObdVarEntryNotExist	OBD: object does not contain VarEntry structure.
kEplObdValueTooLow	OBD: specified object value too low.
kEplObdValueTooHigh	OBD: specified object value too high.
kEplObdValueLengthError	OBD: length of specified value does not match to the object.
kEplNmtUnknownCommand	NMT: unknown NMT command specified.
kEplNmtInvalidFramePointer	NMT: invalid pointer to EPL frame specified.
kEplNmtInvalidEvent	NMT: invalid event passed to event process function.
kEplNmtInvalidState	NMT: invalid NMT state.
kEplNmtInvalidParam	NMT: invalid parameters specified on function call.
kEplSdoAsndInvalidNodeId	SDO ASnd layer: invalid node-ID specified.
kEplSdoAsndNoFreeHandle	SDO ASnd layer: no free handle available. Increase value of define EPL_SDO_MAX_CONNECTION_ASND.
kEplSdoAsndInvalidHandle	SDO ASnd layer: invalid handle specified.
kEplSdoUdpMissCb	SDO/UDP: no pointer to callback function specified.
kEplSdoUdpNoSocket	SDO/UDP: socket could be created.
kEplSdoUdpSocketError	SDO/UDP: unspecified error with socket handling.
kEplSdoUdpThreadError	SDO/UDP: error occurred while creating or terminating thread for UDP processing.
kEplSdoUdpNoFreeHandle	SDO/UDP: no free handle available. Increase value of define EPL_SDO_MAX_CONNECTION_UDP.
kEplSdoUdpSendError	SDO/UDP: error while sending datagram.
kEplSdoUdpInvalidHdl	SDO/UDP: invalid handle specified.
kEplSdoSeqMissCb	SDO sequence layer: no pointer to callback

Constant	***Description***
	function specified.
kEplSdoSeqNoFreeHandle	SDO sequence layer: no free handle available. Increase value of define EPL_MAX_SDO_SEQ_CON.
kEplSdoSeqInvalidHdl	SDO sequence layer: invalid handle specified.
kEplSdoSeqUnsupportedProt	SDO sequence layer: unsupported lower layer protocol specified.
kEplSdoSeqNoFreeHistory	SDO sequence layer: no free entry in history available (internal error).
kEplSdoSeqFrameSizeError	SDO sequence layer: size of frame is larger than value of define EPL_MAX_SDO_FRAME_SIZE.
kEplSdoSeqRequestAckNeeded	SDO sequence layer: acknowledge must be requested from communication partner (internal error).
kEplSdoSeqInvalidFrame	SDO sequence layer: invalid frame specified internally.
kEplSdoSeqConnectionBusy	SDO sequence layer: connection is currently busy.
kEplSdoSeqInvalidEvent	SDO sequence layer: invalid event passed to event process function.
kEplSdoComUnsupportedProt	SDO command layer: unsupported lower layer protocol specified.
kEplSdoComNoFreeHandle	SDO command layer: no free handle available. Increase value of define EPL_MAX_SDO_COM_CON.
kEplSdoComInvalidHandle	SDO command layer: invalid handle specified.
kEplSdoComInvalidSendType	SDO command layer: illegal send type specified internally.
kEplSdoComNotResponsible	SDO command layer: current handle is not responsible (wrong direction or wrong transaction-ID).
kEplSdoComHandleExists	SDO command layer: connection to the same node-ID and with same protocol type exists. The handle of this connection is returned.
kEplSdoComHandleBusy	SDO command layer: connection is busy.
kEplSdoComInvalidParam	SDO command layer: invalid parameters specified on function call.
kEplEventUnknownSink	Event modules: unknown event sink specified.
kEplEventPostError	Event modules: error occurred while posting

Constant	***Description***
	event.
kEplTimerInvalidHandle	Timer modules: invalid handle specified.
kEplTimerNoTimerCreated	Timer modules: no timer was created because of an error.
kEplPdoNotExist	PDO: the selected PDO does not exist.
kEplPdoLengthExceeded	PDO: the length of the PDO mapping exceeds the current payload limit.
kEplPdoGranularityMismatch	PDO: the object is mapped to a bit offset or with a bit length which is not aligned on byte boundaries.
kEplPdoInitError	PDO: an error occurred while initializing the PDO module.
kEplPdoConfWhileEnabled	PDO: the PDO configuration cannot be changed while the corresponding PDO is enabled.
kEplPdoErrorMapp	PDO: the PDO mapping is invalid.
kEplPdoVarNotFound	PDO: the referenced object in a PDO mapping does not exist.
kEplPdoSizeMismatch	PDO: the bit size of the object mapping is larger than or unequal to the size of the referenced object.
kEplPdoTooManyTxPdos	PDO: too many TPDOs are defined in the OD. Pure CNs only supports one TPDO.
kEplPdoInvalidObjIndex	PDO: the OD callback function EplPdouCbObdAccess() is used for an invalid object index in the OD.
kEplPdoTooManyPdos	PDO: too many PDOs are defined in the OD.
kEplCfgMaConfigError	Configuration manager: error while configuring CN (SDO abort).
kEplCfgMaSdocTimeOutError	Configuration manager: SDO timeout while configuring CN
kEplCfgMaInvalidDcf	Configuration manager: invalid DCF specified.
kEplCfgMaUnsupportedDcf	Configuration manager: currently unsupported DCF type specified.
kEplCfgMaConfigWithErrors	Configuration manager: configuration of CN finished with minor errors.
kEplCfgMaNoFreeConfig	Configuration manager: no free entry in internal array. Increase value of define EPL_CFGMA_MAX_SDO_CLIENTS.
kEplCfgMaNoConfigData	Configuration manager: no configuration data

Constant	***Description***
	(DCF) for specified CN available.
kEplCfgMaUnsuppDatatypeDcf	Configuration manager: unsupported data type in DCF.
kEplApiTaskDeferred	EPL API layer: the requested operation was deferred and the event callback function is called when it has finished.
kEplApiInvalidParam	EPL API layer: invalid parameter specified on function call.

Table 4: Constants of enumerated type tEplKernel

2.2.3 tEplMsgType

The enumerated type tEplMsgType represents the EPL frame types.

```
typedef enum
{
    kEplMsgTypeNonEpl = 0x00,
    kEplMsgTypeSoc    = 0x01,
    kEplMsgTypePreq   = 0x03,
    kEplMsgTypePres   = 0x04,
    kEplMsgTypeSoa    = 0x05,
    kEplMsgTypeAsnd   = 0x06,

} tEplMsgType;
```

Constant	***Description***
kEplMsgTypeNonEpl	Non-EPL frame
kEplMsgTypeSoc	EPL frame SoC (Start of Cyclic)
kEplMsgTypePreq	EPL frame PReq (Poll Request)
kEplMsgTypePres	EPL frame PRes (Poll Response)
kEplMsgTypeSoa	EPL frame SoA (Start of Asynchronous)
kEplMsgTypeAsnd	EPL frame ASnd (Asynchronous Send)

Table 5: Constants of enumerated type tEplMsgType

2.3 Functions

2.3.1 EPL API Layer

The EPL API Layer is the interface for the application to the EPL stack. It initializes and configures the different modules of the EPL stack.

2.3.1.1 Event callback function tEplApiCbEvent

Syntax:

```
#include <Epl.h>
typedef tEplKernel (PUBLIC ROM* tEplApiCbEvent) (
            tEplApiEventType                EventType_p,
            tEplApiEventArg*                pEventArg_p,
            void GENERIC*                   pUserArg_p);
```

Parameters:

EventType_p	event type (see Table 6)
pEventArg_p:	Pointer to a union containing additional arguments for the specified event type (see Table 7). It is never a null pointer, but the union pointed to does not contain meaningful information in every case. It depends on EventType_p (see Table 6).
pUserArg_p:	Pointer to a user definable argument

Return:

kEplSuccessful	The function was executed without error.
kEplApiReject	The application wants the EPL stack to defer the subsequent task.
kEplShutdown	Depending on the target platform, the EPL stack will be shut down or this return code will be ignored and treated as kEplSuccessful.

Other return codes will abort the current action in some cases and may cause critical errors.

Description:

Functions of this type can be used as event callback function. This function will be called whenever an event occurs which might be interesting to the application. Depending on the target platform this function may be called simultaneously in different process contexts.

2.3.1.1.1 tEplApiEventType

```
typedef enum
{
    kEplApiEventUserDef         = 0x00,
    kEplApiEventNmtStateChange = 0x10,
    kEplApiEventCriticalError  = 0x12,
    kEplApiEventWarning        = 0x13,
    kEplApiEventNode           = 0x20,
    kEplApiEventBoot           = 0x21,
    kEplApiEventSdo            = 0x62,
    kEplApiEventObdAccess      = 0x69,
    kEplApiEventLed            = 0x70,

} tEplApiEventType;
```

Constant	***Description***
kEplApiEventUserDef	User-defined event. The member m_pUserArg of the argument union is valid. This is issued by the function EplApiPostUserEvent() and can be used for synchronization purposes.
kEplApiEventNmtStateChange	NMT state change event. The member m_NmtStateChange of the argument union is valid. If kEplApiReject is returned the subsequent NMT state will not be entered. In this case the application is in charge of executing the appropriate NMT commands.
kEplApiEventCriticalError	Critical error. The member m_InternalError of the argument union is valid. When this event occurs the NMT state machine will be switched of with NMT event kEplNmtEventCriticalError. The application may restart the NMT state machine afterwards, but it is unlikely that the EPL stack will run stably, because often this critical error or the source of it is a configuration error and not a run-time error.
kEplApiEventWarning	Warning. The member m_InternalError of the argument union is valid. The warning may be a

Constant	***Description***
	run-time error, which should be logged into an error log for further diagnostics. In any case the EPL stack proceeds.
kEplApiEventNode	Node event on MN. The member m_Node of the argument union is valid. The state of the specified node has changed.
kEplApiEventBoot	Boot event on MN. The member m_Boot of the argument union is valid. The MN reached the specified state in the boot-up process.
kEplApiEventSdo	SDO transfer finished. The member m_Sdo of the argument union is valid.
kEplApiEventObdAccess	OBD is being accessed. The member m_EplObdCbParam of the argument union is valid.
kEplApiEventLed	Status and error LED event. The member m_Led of the argument union is valid. This event allows the application to perfrom the signalling of the status and error LED according to [1].

Table 6: Constants for enumerated type tEplApiEventType

2.3.1.1.2 tEplApiEventArg

```
typedef union
{
    void*                       m_pUserArg;
    tEplEventNmtStateChange m_NmtStateChange;
    tEplEventError              m_InternalError;
    tEplSdoComFinished          m_Sdo;
    tEplObdCbParam              m_ObdCbParam;
    tEplApiEventNode            m_Node;
    tEplApiEventBoot            m_Boot;
    tEplApiEventLed             m_Led;

} tEplApiEventArg;
```

Member	***Description***
m_pUserArg	User-defined argument pointer.
m_NmtStateChange	Event from module NMT (valid on kEplApiEventNmtStateChange). See Table 8.

Member	***Description***
m_InternalError	Error from within EPL stack (valid on kEplApiEventCriticalError and kEplApiEventWarning). See Table 11.
m_Sdo	SDO finished (valid on kEplApiEventSdo).
m_ObdCbParam	Parameters for callback function from access to the local OD (valid on kEplApiEventObdAccess). See Table 17.
m_Node	Information about the node event (MN only). See Table 19.
m_Boot	Information about the boot event (MN only).
m_Led	Information about changes to the status resp. error LED.

Table 7: Members of union tEplApiEventArg

The following structures are member elements of the union tEplApiEventArg, i.e. pEventArg_p.

2.3.1.1.3 tEplEventNmtStateChange

```
typedef struct
{
    tEplNmtState    m_NewNmtState;
    tEplNmtEvent    m_NmtEvent;

} tEplEventNmtStateChange;
```

Member	***Description***
m_NewNmtState	New NMT state (see Table 9).
m_NmtEvent	NMT event that caused the NMT state change (see Table 10).

Table 8: Members of structure tEplEventNmtStateChange

The NMT states of EPL are represented by the enumerated type tEplNmtState. Some states require a special reaction from the application and/or the EPL stack, but others represent only a state where certain action may or may not be executed. The manual L-1108 “Introduction into Ethernet POWERLINK Protocol Stack” contains sample code for this case.

```
typedef enum
{
    kEplNmtGsOff                     = 0x0000,
    kEplNmtGsInitialising            = 0x0019,
    kEplNmtGsResetApplication        = 0x0029,
    kEplNmtGsResetCommunication      = 0x0039,
    kEplNmtGsResetConfiguration      = 0x0079,
    kEplNmtCsNotActive               = 0x011C,
    kEplNmtCsPreOperational1         = 0x011D,
    kEplNmtCsStopped                 = 0x014D,
    kEplNmtCsPreOperational2         = 0x015D,
    kEplNmtCsReadyToOperate          = 0x016D,
    kEplNmtCsOperational             = 0x01FD,
    kEplNmtCsBasicEthernet           = 0x011E,
    kEplNmtMsNotActive               = 0x021C,
    kEplNmtMsPreOperational1         = 0x021D,
    kEplNmtMsPreOperational2         = 0x025D,
    kEplNmtMsReadyToOperate          = 0x026D,
    kEplNmtMsOperational             = 0x02FD,
    kEplNmtMsBasicEthernet           = 0x021E

} tEplNmtState;
```

Constant	***Description***
kEplNmtGsOff	Generic NMT state NMT_GS_OFF.
kEplNmtGsInitialising	Generic NMT state NMT_GS_INITIALISING.
kEplNmtGsResetApplication	Generic NMT state NMT_GS_RESET_APPLICATION. The manufacturer-specific and device profile OD parts are reset to defaults.
kEplNmtGsResetCommunication	Generic NMT state NMT_GS_RESET_COMMUNICATION. The communication profile OD part is reset to defaults. Additionally, the OD is updated from initialization parameters.
kEplNmtGsResetConfiguration	Generic NMT state NMT_GS_RESET_CONFIGURATION. The configuration parameters of the DLL module are updated from OD.
kEplNmtCsNotActive	CN NMT state NMT_CS_NOT_ACTIVE.
kEplNmtCsPreOperational1	CN NMT state NMT_CS_PRE_OPERATIONAL_1.
kEplNmtCsStopped	CN NMT state NMT_CS_STOPPED.

Constant	***Description***
kEplNmtCsPreOperational2	CN NMT state NMT_CS_PRE_OPERATIONAL_2.
kEplNmtCsReadyToOperate	CN NMT state NMT_CS_READY_TO_OPERATE.
kEplNmtCsOperational	CN NMT state NMT_CS_OPERATIONAL.
kEplNmtCsBasicEthernet	CN NMT state NMT_CS_BASIC_ETHERNET.
kEplNmtMsNotActive	MN NMT state NMT_MS_NOT_ACTIVE.
kEplNmtMsPreOperational1	MN NMT state NMT_MS_PRE_OPERATIONAL_1.
kEplNmtMsPreOperational2	MN NMT state NMT_MS_PRE_OPERATIONAL_2.
kEplNmtMsReadyToOperate	MN NMT state NMT_MS_READY_TO_OPERATE.
kEplNmtMsOperational	MN NMT state NMT_MS_OPERATIONAL.
kEplNmtMsBasicEthernet	MN NMT state NMT_MS_BASIC_ETHERNET.

Table 9: Constants for enumerated type tEplNmtState

The enumerated type tEplNmtEvent represents all NMT events and commands which alter the NMT state. Only a few of them may be actually used by the application for the function EplApiExecNmtCommand(). But in the event callback function the NMT events tell you why the NMT state has changed.

```
typedef enum
{
    kEplNmtEventNoEvent              =   0x00,
    kEplNmtEventDllMePresTimeout     =   0x02,
    kEplNmtEventDllMeSocTrig         =   0x05,
    kEplNmtEventDllMeSoaTrig         =   0x06,
    kEplNmtEventDllCeSoc             =   0x07,
    kEplNmtEventDllCePreq            =   0x08,
    kEplNmtEventDllCePres            =   0x09,
    kEplNmtEventDllCeSoa             =   0x0A,
    kEplNmtEventDllCeAsnd            =   0x0B,
    kEplNmtEventDllCeFrameTimeout    =   0x0C,
    kEplNmtEventSwReset              =   0x10,
    kEplNmtEventResetNode            =   0x11,
    kEplNmtEventResetCom             =   0x12,
    kEplNmtEventResetConfig          =   0x13,
    kEplNmtEventEnterPreOperational2=   0x14,
    kEplNmtEventEnableReadyToOperate=   0x15,
    kEplNmtEventStartNode           =   0x16,
    kEplNmtEventStopNode            =   0x17,
    kEplNmtEventEnterResetApp       =   0x20,
    kEplNmtEventEnterResetCom       =   0x21,
    kEplNmtEventInternComError      =   0x22,
    kEplNmtEventEnterResetConfig    =   0x23,
    kEplNmtEventEnterCsNotActive    =   0x24,
    kEplNmtEventEnterCsNotActive    =   0x25,
    kEplNmtEventTimerBasicEthernet  =   0x26,
    kEplNmtEventTimerMsPreOp1       =   0x27,
    kEplNmtEventNmtCycleError       =   0x28,
    kEplNmtEventTimerMsPreOp2       =   0x29,
    kEplNmtEventAllMandatoryCNIdent =   0x2A,
    kEplNmtEventEnterReadyToOperate =   0x2B,
    kEplNmtEventEnterMsOperational  =   0x2C,
    kEplNmtEventSwitchOff           =   0x2D,
    kEplNmtEventCriticalError       =   0x2E,

} tEplNmtEvent;
```

Constant	***Description***
kEplNmtEventNoEvent	No event occurred, which is very unlikely ☺.
kEplNmtEventDllMePresTimeout	DLL MN: PRes timed out
kEplNmtEventDllMeSocTrig	DLL MN: SoC triggered
kEplNmtEventDllMeSoaTrig	DLL MN: SoA triggered
kEplNmtEventDllCeSoc	DLL CN: SoC received
kEplNmtEventDllCePreq	DLL CN: PReq received
kEplNmtEventDllCePres	DLL CN: PRes received
kEplNmtEventDllCeSoa	DLL CN: SoA received
kEplNmtEventDllCeAsnd	DLL CN: ASnd received
kEplNmtEventDllCeFrameTimeout	DLL CN: arbitrary EPL frame timed out
kEplNmtEventSwReset	External NMT command: software reset, i.e. enter NMT_GS_INITIALISING. The application may issue this event. It must issue this event after calling EplApiInitialize() to start the NMT state machine.
kEplNmtEventResetNode	External NMT command: reset application, i.e. enter NMT_GS_RESET_APPLICATION. The application may issue this event if necessary.
kEplNmtEventResetCom	External NMT command: reset communication, i.e. enter NMT_GS_RESET_COMMUNICATION.

Constant	***Description***
	The application may issue this event if necessary.
kEplNmtEventResetConfig	External NMT command: reset configuration, i.e. enter NMT_GS_RESET_CONFIGURATION. The application may issue this event if necessary.
kEplNmtEventEnterPreOperational2	External NMT command: enter NMT_CS_PRE_OPERATIONAL_2. This command may be issued only by the MN.
kEplNmtEventEnableReadyToOperate	External NMT command: enter NMT_CS_READY_TO_OPERATE if application approved it. This command may be issued only by the MN.
kEplNmtEventStartNode	External NMT command: enter NMT_CS_OPERATIONAL. This command may be issued only by the MN.
kEplNmtEventStopNode	External NMT command: enter NMT_CS_STOPPED. This command may be issued only by the MN.
kEplNmtEventEnterResetApp	Internal NMT command: reset application, i.e. enter NMT_GS_RESET_APPLICATION. The application must issue this event in kEplNmtGsInitialising if it returned kEplApiReject.
kEplNmtEventEnterResetCom	Internal NMT command: reset communication, i.e. enter NMT_GS_RESET_COMMUNICATION. The application must issue this event in kEplNmtGsResetApplication if it returned kEplApiReject.
kEplNmtEventInternComError	Internal NMT command: reset communication, i.e. enter NMT_GS_RESET_COMMUNICATION. The EPL stack issues this event if an internal communication error occurred which may be cured by reset communication.
kEplNmtEventEnterResetConfig	Internal NMT command: reset configuration, i.e. enter NMT_GS_RESET_CONFIGURATION. The application must issue this event in kEplNmtGsResetCommunication if it returned kEplApiReject.
kEplNmtEventEnterCsNotActive	Internal NMT command: enter NMT state not active, i.e. NMT_CS_NOT_ACTIVE. The application must issue this event in kEplNmtGsResetConfiguration if it returned

Constant	***Description***
	kEplApiReject and wants to act as CN.
kEplNmtEventEnterMsNotActive	Internal NMT command: enter NMT state not active, i.e. NMT_MS_NOT_ACTIVE. The application must issue this event in kEplNmtGsResetConfiguration if it returned kEplApiReject and wants to act as MN.
kEplNmtEventTimerBasicEthernet	Internal timer event to enter NMT state basic ethernet, i.e. either NMT_CS_BASIC_ETHERNET or NMT_MS_BASIC_ETHERNET depending on previous state.
kEplNmtEventTimerMsPreOp1	Internal timer event to enter NMT state NMT_MS_PRE_OPERATIONAL_1.
kEplNmtEventNmtCycleError	Internal NMT command: enter NMT_CS_PRE_OPERATIONAL_1. The error handler issues this event if it detects an EPL cycle error.
kEplNmtEventTimerMsPreOp2	Internal timer event to enter NMT state NMT_MS_PRE_OPERATIONAL_2.
kEplNmtEventAllMandatoryCNIdent	Internal NMT command: enter NMT_MS_PRE_OPERATIONAL_2 when all mandatory CNs are identified.
kEplNmtEventEnterReadyToOperate	Internal NMT command: enter either NMT_CS_READY_TO_OPERATE if MN approved it or NMT_MS_READY_TO_OPERATE if MN is active. The application must issue this event in kEplNmtCsPreOperational2 if it returned kEplApiReject.
kEplNmtEventMsOperational	Internal NMT command: enter NMT_MS_OPERATIONAL.
kEplNmtEventSwitchOff	Internal NMT command: enter NMT_GS_OFF. The application must issue this event before calling EplApiShutdown() to stop the NMT state machine.
kEplNmtEventCriticalError	Internal NMT command: enter NMT_GS_OFF. The EPL stack issues this event if an internal error occurred which is unlikely to be cured by reset.

Table 10: Constants for enumerated type tEplNmtEvent

2.3.1.1.4 tEplEventError

The structure tEplEventError describes an error event and where it comes from.

```
typedef struct
{
    tEplEventSource m_EventSource;
    tEplKernel      m_EplError;
    union
    {
        BYTE                    m_bArg;
        DWORD                   m_dwArg;
        tEplEventSource         m_EventSource;
        tEplEventObdError       m_ObdError;

    } m_Arg;

} tEplEventError;
```

Member	***Description***
m_EventSource	Source module of the error, which determines the valid member of the union m_Arg (see Table 12).
m_EplError	Internal EPL stack error code (see Table 4)
m_Arg.m_bArg	BYTE argument
m_Arg.m_dwArg	DWORD argument
m_Arg.m_EventSource	Originating source module (valid on kEplEventSourceEventk and kEplEventSourceEventu). See Table 12.
m_Arg.m_ObdError	Failing entry of OD (valid on kEplEventSourceObdk and kEplEventSourceObdu). See Table 13.

Table 11: Members of structure tEplEventError

```
typedef enum
{
    kEplEventSourceDllk         = 0x01,
    kEplEventSourceNmtk         = 0x02,
    kEplEventSourceObdk         = 0x03,
    kEplEventSourcePdok         = 0x04,
    kEplEventSourceTimerk       = 0x05,
    kEplEventSourceEventk       = 0x06,
    kEplEventSourceSyncCb       = 0x07,
    kEplEventSourceErrk         = 0x08,
    kEplEventSourceDllu         = 0x10,
    kEplEventSourceNmtu          = 0x11,
    kEplEventSourceNmtCnu        = 0x12,
    kEplEventSourceNmtMnu        = 0x13,
    kEplEventSourceObdu          = 0x14,
    kEplEventSourceSdoUdp        = 0x15,
    kEplEventSourceSdoAsnd       = 0x16,
    kEplEventSourceSdoAsySeq     = 0x17,
    kEplEventSourceSdoCom        = 0x18,
    kEplEventSourceTimeru        = 0x19,
    kEplEventSourceCfgMau        = 0x1A,
    kEplEventSourceEventu        = 0x1B,
    kEplEventSourceEplApi        = 0x1C

} tEplEventSource;
```

Constant	***Description***
kEplEventSourceDllk	DLL module in EPL kernel part
kEplEventSourceNmtk	NMT module in EPL kernel part
kEplEventSourceObdk	OBD module in EPL kernel part
kEplEventSourcePdok	PDO module in EPL kernel part
kEplEventSourceTimerk	Timer module in EPL kernel part
kEplEventSourceEventk	Event module in EPL kernel part
kEplEventSourceSyncCb	Sync callback function
kEplEventSourceErrk	Error handler module in EPL kernel part
kEplEventSourceDllu	DLL module in EPL user part
kEplEventSourceNmtu	NMT module in EPL user part
kEplEventSourceNmtCnu	NMT CN module in EPL user part
kEplEventSourceNmtMnu	NMT MN module in EPL user part
kEplEventSourceObdu	OBD module in EPL user part
kEplEventSourceSdoUdp	SDO UDP protocol layer
kEplEventSourceSdoAsnd	SDO ASnd protocol layer
kEplEventSourceSdoAsySeq	SDO sequence layer
kEplEventSourceSdoCom	SDO command layer
kEplEventSourceTimeru	Timer module in EPL user part
kEplEventSourceCfgMau	Configuration Manager module in EPL user part
kEplEventSourceEventu	Event module in EPL user part
kEplEventSourceEplApi	EPL API Layer

Table 12: Constants for enumerated type tEplEventSource

```
typedef struct
{
    unsigned int    m_uiIndex;
    unsigned int    m_uiSubIndex;

} tEplEventObdError;
```

Member	***Description***
m_uiIndex	Object dictionary index.
m_uiSubIndex	Object dictionary sub index.

Table 13: Members of structure tEplEventObdError

2.3.1.1.5 tEplSdoComFinished

The structure tEplSdoComFinished contains the information of the finished SDO transfer started by **EplApiReadObject()** or **EplApiWriteObject()**.

```
typedef struct
{
    tEplSdoComConHdl    m_SdoComConHdl;
    tEplSdoComConState  m_SdoComConState;
    DWORD               m_dwAbortCode;
    tEplSdoAccessType   m_SdoAccessType;
    unsigned int        m_uiNodeId;
    unsigned int        m_uiTargetIndex;
    unsigned int        m_uiTargetSubIndex;
    unsigned int        m_uiTransferredByte;
    void*               m_pUserArg;

} tEplSdoComFinished;
```

Member	***Description***
m_SdoComConHdl	SDO command layer connection handle (see section 2.3.1.6).
tEplSdoComConState	State of the transfer (see Table 15).
m_dwAbortCode	SDO abort code.
m_SdoAccessType	Type of SDO access (see Table 16).
m_uiNodeId	Target node-ID.
m_uiTargetIndex	OD index which was accessed.
m_uiTargetSubIndex	OD sub index which was accessed.
m_uiTransferredByte	Number of bytes transferred.

Member	***Description***
m_pUserArg	Pointer to user definable argument (see section 2.3.1.6).

Table 14: Members of structure tEplSdoComFinished

```
typedef enum
{
    kEplSdoComTransferNotActive         =   0x00,
    kEplSdoComTransferRunning           =   0x01,
    kEplSdoComTransferTxAborted         =   0x02,
    kEplSdoComTransferRxAborted         =   0x03,
    kEplSdoComTransferFinished          =   0x04,
    kEplSdoComTransferLowerLayerAbort   =   0x05

} tEplSdoComConState;
```

Constant	***Description***
kEplSdoComTransferNotActive	SDO transfer is not active. Not applicable in structure tEplSdoComFinished.
kEplSdoComTransferRunning	SDO transfer is still running. Not applicable in structure tEplSdoComFinished.
kEplSdoComTransferTxAborted	SDO transmission aborted. The abort code m_dwAbortCode contains more details.
kEplSdoComTransferRxAborted	SDO reception aborted. The abort code m_dwAbortCode contains more details.
kEplSdoComTransferFinished	SDO transfer finished successfully.
kEplSdoComTransferLowerLayerAbort	SDO transfer aborted by a lower layer, e.g. the SDO sequence layer.

Table 15: Constants for enumerated type tEplSdoComConState

```
typedef enum
{
    kEplSdoAccessTypeRead   = 0x00,
    kEplSdoAccessTypeWrite  = 0x01

} tEplSdoAccessType;
```

Constant	***Description***
kEplSdoAccessTypeRead	SDO read access.

Constant	***Description***
kEplSdoAccessTypeWrite	SDO write access.

Table 16: Constants for enumerated type tEplSdoAccessType

2.3.1.1.6 tEplObdCbParam

The structure tEplObdCbParam contains the information of the local OD access that is being made.

```
typedef struct
{
    tEplObdEvent    m_ObdEvent;
    unsigned int    m_uiIndex;
    unsigned int    m_uiSubIndex;
    void*           m_pArg;
    DWORD           m_dwAbortCode;

} tEplObdCbParam;
```

Member	***Description***
m_ObdEvent	Object access event (see Table 18).
m_uiIndex	Object index.
m_uiSubIndex	Object sub index.
m_pArg	Pointer to argument which type depends on access event.
m_dwAbortCode	SDO abort code

Table 17: Members of structure tEplObdCbParam

```
typedef enum
{
    kEplObdEvCheckExist            = 0x06,
    kEplObdEvPreRead               = 0x00,
    kEplObdEvPostRead              = 0x01,
    kEplObdEvWrStringDomain        = 0x07,
    kEplObdEvInitWrite             = 0x04,
    kEplObdEvPreWrite              = 0x02,
    kEplObdEvPostWrite             = 0x03,

} tEplObdEvent;
```

Constant	***Description***
kEplObdEvCheckExist	Checking if object exists (m_pArg == NULL).

Constant	***Description***
kEplObdEvPreRead	Before reading the object. m_pArg points to the source data buffer in OD.
kEplObdEvPostRead	After reading the object. m_pArg points to the destination data buffer from the caller.
kEplObdEvWrStringDomain	Changing string/domain data pointer or size. m_pArg points to the structure tEplObdVStringDomain, which may be altered by application.
kEplObdEvInitWrite	Initializing writing an object. m_pArg points to value of type tEplObdSize, which represents the number of bytes that will be written (may be altered by application).
kEplObdEvPreWrite	Before writing an object. m_pArg points to the source data buffer from the caller.
kEplObdEvPostWrite	After writing an object. m_pArg points to the destination data buffer in OD.

Table 18: Constants for enumerated type tEplObdEvent

2.3.1.1.7 tEplApiEventNode

The following types are valid on MN only.

The structure tEplApiEventNode contains the information about a single CN in the boot-up process of the MN.

```
typedef struct
{
    unsigned int        m_uiNodeId;
    tEplNmtState        m_NmtState;
    tEplNmtNodeEvent    m_NodeEvent;
    WORD                m_wErrorCode;
    BOOL                m_fMandatory;

} tEplApiEventNode;
```

Member	***Description***
m_uiNodeId	Node-ID of the affected CN.
m_NmtState	Current NMT state of the CN (see Table 9).
m_NodeEvent	Specific event of the CN (see Table 20).

Member	***Description***
m_wErrorCode	EPL error code if m_NodeEvent equals kEplNmtNodeEventError (see Table 21).
m_fMandatory	TRUE if CN was configured as mandatory in local OD index 0x1F81 (NMT_NodeAssignment_AU32), otherwise it is an optional CN.

Table 19: Members of structure tEplApiEventNode

```
typedef enum
{
    kEplNmtNodeEventFound       = 0x00,
    kEplNmtNodeEventUpdateSw    = 0x01,
    kEplNmtNodeEventCheckConf   = 0x02,
    kEplNmtNodeEventUpdateConf  = 0x03,
    kEplNmtNodeEventVerifyConf  = 0x04,
    kEplNmtNodeEventReadyToStart= 0x05,
    kEplNmtNodeEventNmtState    = 0x06,
    kEplNmtNodeEventError       = 0x07,

} tEplNmtNodeEvent;
```

Constant	***Description***
kEplNmtNodeEventFound	CN answered to IdentRequest with an IdentResponse frame.
kEplNmtNodeEventUpdateSw	Application shall update the software on the CN (currently not implemented).
kEplNmtNodeEventCheckConf	Application resp. the Configuration Manager shall check and update configuration on CN. If the application returns kEplReject, the MN suspends the boot-up process for this CN until the application triggers the state change (see section 2.3.1.6). Otherwise the MN continues the CN boot-up process automatically with the next state.
kEplNmtNodeEventUpdateConf	Application resp. the Configuration Manager shall update configuration on CN (check was done by NmtMn module, currently not implemented).
kEplNmtNodeEventVerifyConf	Application resp. the Configuration Manager shall verify configuration of CN (currently not implemented).
kEplNmtNodeEventReadyToStart	Issued if EPL_NMTST_NO_STARTNODE set (currently not implemented). Application must call EplNmtMnuSendNmtCommand(kEplNmtCmdStartNode) manually.

Constant	***Description***
kEplNmtNodeEventNmtState	NMT state of CN has changed.
kEplNmtNodeEventError	Error occurred with CN (see m_wErrorCode for details).

Table 20: Constants for enumerated type tEplNmtNodeEvent

Constant	***Value***	***Description***
EPL_E_NO_ERROR	0x0000	No error actually occurred.
EPL_E_DLL_INVALID_FORMAT	0x8241	Format of received frame is invalid. Issued by DLL.
EPL_E_DLL_LOSS_PRES_TH	0x8243	The threshold for loss of PRes frames was reached according to objects 0x1C09 DLL_MNCNLossPResThreshold_AU32 and 0x1C08 DLL_MNCNLossPResThrCnt_AU32.
EPL_E_NMT_BPO1_DEVICE_TYPE	0x8422	The specified CN has the wrong device type according to object 0x1F84 NMT_MNDeviceTypeIdList_AU32.
EPL_E_NMT_BPO1_CF_VERIFY	0x8428	Verification of configuration of the specified CN failed in boot-up process.
EPL_E_NMT_BPO2	0x8430	Mandatory CN failed in boot step 2.
EPL_E_NMT_WRONG_STATE	0x8480	The specified CN has the wrong NMT state.

Table 21: Constants for EPL error code

2.3.1.1.8 tEplApiEventBoot

The following types are valid on MN only.

The structure tEplApiEventBoot contains the information about an event concerning the entire boot-up process of the MN.

```
typedef struct
{
    tEplNmtState        m_NmtState;
    tEplNmtBootEvent    m_BootEvent;
    WORD                m_wErrorCode;
} tEplApiEventBoot;
```

Member	***Description***
m_NmtState	Current local NMT state (see Table 9).
m_BootEvent	Specific event of the boot-up process (see Table 23).
m_wErrorCode	EPL error code if m_BootEvent equals kEplNmtBootEventError (see Table 21).

Table 22: Members of structure tEplApiEventBoot

```
typedef enum
{
    kEplNmtBootEventBootStep1Finish = 0x00,
    kEplNmtBootEventBootStep2Finish = 0x01,
    kEplNmtBootEventCheckComFinish  = 0x02,
    kEplNmtBootEventOperational     = 0x03,
    kEplNmtBootEventError           = 0x04,

} tEplNmtBootEvent;
```

Constant	***Description***
kEplNmtBootEventBootStep1Finish	Boot step 1 has finished, NMT state NMT_MS_PRE_OPERATIONAL2 can be entered. If the application returns kEplReject, it is in charge of triggering this state change, otherwise it is done automatically.
kEplNmtBootEventBootStep2Finish	Boot step 2 has finished, NMT state NMT_MS_READY_TO_OPERATE can be entered. If the application returns kEplReject, it is in charge of triggering this state change, otherwise it is done automatically.
kEplNmtBootEventCheckComFinish	Step "Check communication" has finished, CNs can be started. If the application returns kEplReject, it is in charge of triggering this state change, otherwise it is done automatically.
kEplNmtBootEventOperational	All mandatory CNs entered NMT_CS_OPERATIONAL, NMT state NMT_MS_OPERATIONAL can be entered. If the application returns kEplReject, it is in charge of triggering this state change, otherwise it is done automatically.
kEplNmtBootEventError	Boot-up process was halted because of an error.

Table 23: Constants for enumerated type tEplNmtBootEvent

2.3.1.1.9 tEplApiEventLed

The structure tEplApiEventLed contains change events for the status resp. error LED. It allows the application to change the status and error LED on the device according to the specification [1].

```
typedef struct
{
    tEplLedType         m_LedType;
    BOOL                m_fOn;

} tEplApiEventLed;
```

Member	***Description***
m_LedType	Type of the LED (e.g. Status or Error).
m_fOn	State of the LED (e.g. on or off).

Table 24: Members of structure tEplApiEventLed

```
typedef enum
{
    kEplLedTypeStatus   = 0x00,
    kEplLedTypeError    = 0x01,

} tEplLedType;
```

Constant	***Description***
kEplLedTypeStatus	State of the status LED shall be changed.
kEplLedTypeError	State of the error LED shall be changed.

Table 25: Constants for enumerated type tEplLedType

2.3.1.2 Sync callback function tEplApiCbSync

Syntax:

#include <Epl.h>

```
typedef tEplKernel (PUBLIC* tEplSyncCb) (void);
```

Parameters:

none

Return:

kEplSuccessful	The function was executed without error and the transmit PDOs shall be marked valid, i.e. the flag READY shall be set.
kEplReject	The function was executed without error, but the transmit PDOs shall not be marked valid, i.e. the flag READY shall not be set.

Description:

Functions of this type can be used as sync callback function. This function will be called in NMT states PREOPERATIONAL2 or above whenever the sync event occurs. On MN this is when SoC frame is sent and on CN this is when SoC frame is received or the reception of it is anticipated.

This function is the only place where the process variables may be accessed safely, i.e. without interfering with the PDO processing. Normally, the application reads the sensors and sets the actuators in this function synchronously with all other nodes in the network.

The application shall return from this function as fast as possible.

Note:

The sync callback function is not called by the current EPL API Layer in Linux userspace.

2.3.1.3 Function EplApiInitialize()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiInitialize(tEplApiInitParam* pInitParam_p);
```

Parameters:

pInitParam_p:	Pointer to the initialization structure (see Table 26).

Return:

kEplSuccessful	The function was executed without error.
kEplApiInvalidParam	The function was called with invalid parameters, e.g. no event callback function was specified.
kEplNoResource	The function or a called function was not able to create a system dependant resource like a shared memory buffer.

The function may pass return codes from EPL stack modules.

Description:

The function initializes an EPL stack instance.

The elements of the parameter structure tEplApiInitParam are supposed to be specified in platform byte order. For example the most significant bits of m_dwIpAddress specify the network part of the IP address, which shall be 0xC0A86400 (192.168.100.0) according to the standard.

```
typedef struct
{
    unsigned int        m_uiSizeOfStruct;
    BOOL                m_fAsyncOnly;
    unsigned int        m_uiNodeId;
    BYTE                m_abMacAddress[6];
    DWORD               m_dwFeatureFlags;
    DWORD               m_dwCycleLen;
    unsigned int        m_uiIsochrTxMaxPayload;
    unsigned int        m_uiIsochrRxMaxPayload;
    DWORD               m_dwPresMaxLatency;
    unsigned int        m_uiPreqActPayloadLimit;
    unsigned int        m_uiPresActPayloadLimit;
    DWORD               m_dwAsndMaxLatency;
    unsigned int        m_uiMultiplCycleCnt;
    unsigned int        m_uiAsyncMtu;
    unsigned int        m_uiPrescaler;
    DWORD               m_dwLossOfFrameTolerance;
    DWORD               m_dwWaitSocPreq;
    DWORD               m_dwAsyncSlotTimeout;
    DWORD               m_dwDeviceType;
    DWORD               m_dwVendorId;
    DWORD               m_dwProductCode;
    DWORD               m_dwRevisionNumber;
    DWORD               m_dwSerialNumber;
    QWORD               m_qwVendorSpecificExt1;
    DWORD               m_dwVerifyConfigurationDate;
    DWORD               m_dwVerifyConfigurationTime;
    DWORD               m_dwApplicationSwDate;
    DWORD               m_dwApplicationSwTime;
    DWORD               m_dwIpAddress;
    DWORD               m_dwSubnetMask;
    DWORD               m_dwDefaultGateway;
    BYTE                m_sHostname[32];
    BYTE                m_abVendorSpecificExt2[48];
    char*               m_pszDevName;
    char*               m_pszHwVersion;
    char*               m_pszSwVersion;
    tEplApiCbEvent      m_pfnCbEvent;
    void*               m_pEventUserArg;
    tEplSyncCb          m_pfnCbSync;

} tEplApiInitParam;
```

Parameter	***Description***
m_uiSizeOfStruct	Size of this structure. This will be used in future to recognize new parameters.
m_fAsyncOnly	TRUE means the node does not take part in the isochronous phase. It communicates only asynchronously. Reception of PDOs is possible, but no transmission of PDOs.
m_uiNodeID	Local node ID. (0x01 – 0xFE)
m_abMacAddress	Local MAC address
m_dwFeatureFlags	Feature flags in local OD index 0x1F82 (NMT_FeatureFlags_U32)
m_dwCycleLen	Cycle length in [µs] in local OD index 0x1006 (NMT_CycleLen_U32)
m_uiIsochrTxMaxPayload	Maximum isochronous transmit payload in local OD index 0x1F98/1 (IsochrTxMaxPayload_U16)
m_uiIsochrRxMaxPayload	Maximum isochronous receive payload in local OD index 0x1F98/2 (IsochrRxMaxPayload_U16)
m_dwPresMaxLatency	Maximum PRes latency in local OD index 0x1F98/6 (ASndMaxLatency_U32)
m_uiPreqActPayloadLimit	
m_uiPresActPayloadLimit	
m_dwAsndMaxLaten	Maximum ASnd latency in local OD index 0x1F98/3

Parameter	***Description***
cy	(PResMaxLatency_U32)
m_uiMultiplCycleCnt	Multiplexed cycle count in local OD index 0x1F98/7 (MultiplCycleCnt_U8)
m_uiAsyncMtu	Asynchronous MTU in local OD index 0x1F98/8 (AsyncMTU_U16)
m_uiPrescaler	Prescaler in local OD index 0x1F98/9 (Prescaler_U16)
m_dwLossOfFrameTolerance	Loss of frame tolerance in [ns] in local OD index 0x1C14 (DLL_LossOfFrameTolerance_U32)
m_dwWaitSocPreq	Delay between SoC and first PReq in [ns] (MN only) stored in local OD index 0x1F8A/1 (NMT_MNCycleTiming_REC.WaitSoCPReq_U32)
m_dwAsyncSlotTimeout	Timeout of the asynchronous slot in [ns] (MN only) stored in local OD index 0x1F8A/2 (NMT_MNCycleTiming_REC.AsyncSlotTimeout_U32)
m_dwDeviceType	Device profile in local OD index 0x1000/0 (NMT_DeviceType_U32)
m_dwVendorId	Vendor ID in local OD index 0x1018/1 (NMT_IdentityObject_REC.VendorId_U32)
m_dwProductCode	Product code in local OD index 0x1018/2 (NMT_IdentityObject_REC.ProductCode_U32)
m_dwRevisionNumber	Revision number in local OD index 0x1018/3 (NMT_IdentityObject_REC.RevisionNo_U32)
m_dwSerialNumber	Serial number in local OD index 0x1018/4 (NMT_IdentityObject_REC.SerialNo_U32)
m_qwVendorSpecificExt1	Vendor specific extensions 1 in IdentResponse
m_dwVerifyConfigurationDate	CFM_VerifyConfiguration_REC.ConfDate_U32
m_dwVerifyConfigurationTime	CFM_VerifyConfiguration_REC.ConfTime_U32
m_dwApplicationSwDate	Application software date
m_dwApplicationSwTime	Application software time
m_dwIpAddress	IP address of local node
m_dwSubnetMask	Subnet mask of local node
m_dwDefaultGateway	Default gateway of local node
m_sHostname	DNS host name of local node (maximum length: 32

Parameter	***Description***
	characters, allowed characters: 0-9, A-Z, a-z, -)
m_abVendorSpecific Ext2	Vendor specific extension 2 for IdentResponse
m_pszDevName	Pointer to string with device name in local OD index 0x1008/0. It shall be not longer than EPL_MAX_ODSTRING_SIZE including the terminating null character.
m_pszHwVersion	Pointer to string with hardware version in local OD index 0x1009/0. It shall be not longer than EPL_MAX_ODSTRING_SIZE including the terminating null character.
m_pszSwVersion	Pointer to string with software version in local OD index 0x100A/0. It shall be not longer than EPL_MAX_ODSTRING_SIZE including the terminating null character.
m_pfnCbEvent	Pointer to application’s event callback function (see section 2.3.1.1).
m_pEventUserArg	Pointer to a user definable argument of the event callback function
m_pfnCbSync	Pointer to the application’s sync callback function (see section 0).

Table 26: Parameters of the structure tEplApiInitParam

2.3.1.4 Function EplApiShutDown()

Syntax:

#include <Epl.h>

tEplKernel PUBLIC EplApiShutdown(void);

Parameters:

none

Return:

kEplSuccessful	The function was executed without error.

Description:

The function deletes this instance of EPL stack including the Ethernet driver. It is the responsibility of the application to switch off the NMT state machine before calling this function by executing the NMT command kEplNmtEventSwitchOff and waiting for NMT event kEplNmtGsOff.

2.3.1.5 Function EplApiExecNmtCommand()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiExecNmtCommand(tEplNmtEvent NmtEvent_p);
```

Parameters:

NmtEvent_p:	NMT command which shall be executed (see Table 10).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function executes a NMT command, i.e. post the NMT command/event to the Nmtk module. NMT commands which are not appropriate in the current NMT state are silently ignored. Please keep in mind that the NMT state may change until the NMT command is actually executed.

2.3.1.6 Function EplApiMnTriggerStateChange()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiMnTriggerStateChange(
            unsigned int                    uiNodeId_p,
            tEplNmtNodeCommand              NodeCommand_p);
```

Parameters:

uiNodeId_p	Node-ID of the CN.

NodeCommand_p:	Node command which shall be executed for the specified CN (see Table 27).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function triggers a state change of the CN boot-up process for the specified node. It is only available on MN. This function has to be called by the application to resume the CN boot-up process if it suspended this process on event kEplApiEventNode (see Table 19) with the return code kEplReject.

```
typedef enum
{
    kEplNmtNodeCommandBoot      = 0x01,
    kEplNmtNodeCommandSwOk      = 0x02,
    kEplNmtNodeCommandSwUpdated = 0x03,
    kEplNmtNodeCommandConfOk    = 0x04,
    kEplNmtNodeCommandConfReset = 0x05,
    kEplNmtNodeCommandConfErr   = 0x06,
    kEplNmtNodeCommandStart     = 0x07,

} tEplNmtNodeCommand;
```

Constant	***Description***
kEplNmtNodeCommandBoot	If EPL_NODEASSIGN_START_CN is not set for the CN, it must be issued on event kEplNmtNodeEventFound (see Table 20).
kEplNmtNodeCommandSwOk	An update of the software on the CN is not necessary, so the boot-up process can be continued (currently not implemented).
kEplNmtNodeCommandSwUpdated	The application resp. the Configuration Manager updated the software on the CN successfully (currently not implemented).
kEplNmtNodeCommandConfOk	An update of the configuration of the CN is not necessary, so the boot-up process can be continued.
kEplNmtNodeCommandConfReset	The application resp. the Configuration Manager has updated the configuration on the CN successfully. The MN will send the NMT command reset configuration to the CN to activate the new configuration on the CN and restart the identification process.

Constant	***Description***
kEplNmtNodeCommandConfErr	The application resp. the Configuration Manager failed on updating configuration on the CN.
kEplNmtNodeCommandStart	If EPL_NMTST_NO_STARTNODE is set, it must be issued on event kEplNmtNodeEventReadyToStart (currently not implemented). See Table 20.

Table 27: Constants for enumerated type tEplNmtNodeCommand

2.3.1.7 Function EplApiReadObject()

Syntax:

```
#include <Epl.h>

tEplKernel PUBLIC EplApiReadObject(
            tEplSdoComConHdl*           pSdoComConHdl_p,
            unsigned int                uiNodeId_p,
            unsigned int                uiIndex_p,
            unsigned int                uiSubindex_p,
            void*                       pDstData_le_p,
            unsigned int*               puiSize_p,
            tEplSdoType                 SdoType_p,
            void*                       pUserArg_p);
```

Parameters:

pSdoComConHdl_p	pointer to SDO command layer connection handle (may be NULL on local OD access)
uiNodeId_p:	node ID (0 = local node)
uiIndex_p:	index of object in OD
uiSubindex_p:	sub-index of object in OD
pDstData_le_p:	pointer to data in little endian byte order
puiSize_p:	pointer to size of data, in case of local OD access the size actual read is returned.
SdocType_p:	type of SDO transfer (see Table 28). This parameter does not have any affect on local OD access.
pUserArg_p:	pointer to user definable argument

Return:

kEplSuccessful	The function was executed without error.

kEplApiTaskDeferred	Task was deferred. The application is informed via the event callback function, when the task has finished.
kEplApiInvalidParam	The function was called with an invalid parameters.
kEplSdoComInvalidParam	The function was called with invalid SDO parameters like uiIndex_p == 0 etc.
kEplInvalidNodeId	Invalid node ID was specified.
kEplSdoComHandleBusy	A SDO transfer to this node is running currently.
kEplSdoComNoFreeHandle	All SDO command layer connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO command layer connection handles at compile time.
kEplSdoSeqNoFreeHandle	All SDO sequence layer connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO sequence layer connection handles at compile time.
kEplSdoUdpNoFreeHandle	All SDO UDP protocol connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO UDP protocol connection handles at compile time.
kEplSdoAsndNoFreeHandle	All SDO ASnd protocol connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO ASnd protocol connection handles at compile time.
kEplSdoSeqUnsupportedProt	Unsupported SDO type specified.

Description:

The function reads the specified entry from the OD of the specified node. If this node is a remote node, it performs a SDO transfer, which means this function returns kEplApiTaskDeferred and the application is informed via the event callback function when the task is completed. If the target node is the local node, it accesses directly the local OD and returns the data in little endian byte order.

The event type kEplApiEventSdo signals the completion of this task.

In the current implementation of the EPL stack only one SDO transfer is possible to an arbitrary node via a specific protocol at any time. In the future the EPL stack may support more than one SDO command layer connection via the same sequence layer. But the destination node also needs to support more than one SDO command layer connection via the same sequence layer.

The SDO command layer connection handle will be created automatically. The application is in charge of freeing unneeded SDO command layer connection handles via EplApiFreeSdoChannel().

The SDO types kEplSdoTypeAuto and kEplSdoTypePdo are currently not supported.

```
typedef enum
{
    kEplSdoTypeAuto  =   0x00,
    kEplSdoTypeUdp   =   0x01,
    kEplSdoTypeAsnd  =   0x02,
    kEplSdoTypePdo   =   0x03

} tEplSdoType;
```

Constant	***Description***
kEplSdoTypeAuto	automatically detect SDO transfer type, by executing a NMT IdentRequest for the destination node and read the supported SDO type
kEplSdoTypeUdp	use SDO via UDP
kEplSdoTypeAsnd	use SDO via ASnd frames
kEplSdoTypePdo	use SDO via PDO

Table 28: Constants for enumerated type tEplSdoType

2.3.1.8 Function EplApiWriteObject()

Syntax:

#include <Epl.h>

```
tEplKernel PUBLIC EplApiWriteObject(
            tEplSdoComConHdl*               pSdoComConHdl_p,
            unsigned int                    uiNodeId_p,
            unsigned int                        uiIndex_p,
            unsigned int                        uiSubindex_p,
            void*                               pSrcData_le_p,
            unsigned int                        uiSize_p,
            tEplSdoType                         SdoType_p,
            void*                               pUserArg_p);
```

Parameters:

pSdoComConHdl_p	pointer to SDO command layer connection handle (may be NULL on local OD access)
uiNodeId_p:	node ID (0 = local node)
uiIndex_p:	index of object in OD
uiSubindex_p:	sub-index of object in OD
pSrcData_le_p:	pointer to data in little endian byte order
uiSize_p:	size of data
SdocType_p:	type of SDO transfer (see Table 28). This parameter does not have any affect on local OD access.
pUserArg_p:	pointer to user definable argument

Return:

kEplSuccessful	The function was executed without error.
kEplApiTaskDeferred	Task was deferred. The application is informed via the event callback function, when the task has finished.
kEplApiInvalidParam	The function was called with invalid parameters.
kEplSdoComInvalidParam	The function was called with invalid SDO parameters like uiIndex_p == 0 etc.
kEplInvalidNodeId	Invalid node ID was specified.
kEplSdoComHandleBusy	A SDO transfer to this node is running currently.
kEplSdoComNoFreeHandle	All SDO command layer connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO command layer connection handles at compile time.
kEplSdoSeqNoFreeHandle	All SDO sequence layer connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO sequence layer connection handles at compile time.
kEplSdoUdpNoFreeHandle	All SDO UDP protocol connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try

	it later again. Another solution is to define more SDO UDP protocol connection handles at compile time.
kEplSdoAsndNoFreeHandle	All SDO ASnd protocol connection handles are occupied. The application can call EplApiFreeSdoChannel() for other nodes and / or try it later again. Another solution is to define more SDO ASnd protocol connection handles at compile time.
kEplSdoSeqUnsupportedProt	Unsupported SDO type specified.

Description:

The function writes the specified entry to the OD of the specified node. If this node is a remote node, it performs a SDO transfer, which means this function returns kEplApiTaskDeferred and the application is informed via the event callback function when the task is completed.

For further details see EplApiReadObject().

2.3.1.9 Function EplApiFreeSdoChannel()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiFreeSdoChannel(
            tEplSdoComConHdl*                pSdoComConHdl_p);
```

Parameters:

pSdoComConHdl_p	pointer to SDO command layer connection handle

Return:

kEplSuccessful	The function was executed without error.
kEplSdoComInvalidHandle	Invalid SDO command layer connection handle was specified.

Description:

The function releases the specified SDO command layer connection handle.

2.3.1.10 Function EplApiReadLocalObject()

Syntax:

```
#include <Epl.h>

tEplKernel PUBLIC EplApiReadLocalObject(
            unsigned int                    uiIndex_p,
            unsigned int                    uiSubindex_p,
            void*                           pDstData_p,
            unsigned int*                   puiSize_p);
```

Parameters:

uiIndex_p:	index of object in OD
uiSubindex_p:	sub-index of object in OD
pDstData_p:	pointer to data in platform byte order
puiSize_p:	pointer to size of data buffer, the size actual read is returned.

Return:

kEplSuccessful	The function was executed without error.

Description:

The function reads the specified entry from the local OD

2.3.1.11 Function EplApiWriteLocalObject()

Syntax:

```
#include <Epl.h>

tEplKernel PUBLIC EplApiWriteLocalObject(
            unsigned int                    uiIndex_p,
            unsigned int                    uiSubindex_p,
            void*                           pSrcData_p,
            unsigned int                    uiSize_p);
```

Parameters:

uiIndex_p:	index of object in OD
uiSubindex_p:	sub-index of object in OD
pSrcData_p:	pointer to data in platform byte order
uiSize_p:	size of data

Return:

kEplSuccessful	The function was executed without error.

Description:

The function writes the specified entry to the local OD.

2.3.1.12 Function EplApiLinkObject()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiLinkObject(
            unsigned int                        uiObjIndex_p,
            void*                               pVar_p,
            unsigned int*                       puiVarEntries_p,
            tEplObdSize*                        pEntrySize_p,
            unsigned int                        uiFirstSubindex_p );
```

Parameters:

uiObjIndex_p:	Function defines variables for this object
pVar_p:	Pointer to data memory area for the specified object
puiVarEntries_p:	Pointer to number of entries to be defined.
pEntrySize_p:	Pointer to size of one entry. If it equals 0, the entry size is read from the OD. After return the variable contains the count of bytes used from pVar_p.
uiFirstSubindex_p:	This is the first subindex to be mapped.

Return:

kEplSuccessful	The function was executed without error.
kEplObdIndexNotExist	The specified object index does not exist in OD.
kEplObdSubindexNotExist	The specified sub index does not exist in object index.
kEplObdVarEntryNotExist	The object does not contain a VarEntry structure. The object was not properly configured in ***objdict.h***.

Description:

The function maps an application variable to an entry of the object dictionary. By passing a pointer to an array multiple sub-indices are defined, i.e. mapped, by one function call.

This function may be used to link process variables to the OD so that they can be mapped to PDOs.

In the current implementation this function is not available in the EPL API Layer in Linux userspace, because of the different address spaces of application and EPL stack.

2.3.1.13 Function EplApiProcess()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiProcess(void);
```

Parameters:

none

Return:

kEplSuccessful	The function was executed without error.

Description:

The function is currently only available in EPL API Layer in Linux userspace (file EplApiLinuxUser.c). It assigns CPU time to the EPL stack for processing of events. It waits for events from the EPL stack and calls the event callback function. It should be executed in a separate thread. If the event callback function returns tEplShutdown or the process receives a POSIX signal the function returns.

2.3.1.14 Function EplApiProcessImageSetup()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiProcessImageSetup(void);
```

Parameters:

none

Return:

kEplSuccessful	The function was executed without error.

The function returns the error codes from **EplApiLinkObject**().

Description:

The function sets up the static input and output process image with the respective size configured in ***EplCfg.h*** and links the images as multiple process variables to the OD. The images are linked multiple times but with different types as overlay to the OD. The application is in charge of using the sub indices of the different object indices in the right way.
The static process image provides an easy way to exchange a larger number of process variables between two address spaces like from Linux kernel to userspace and vice versa. Currently, **EplApiProcessImageSetup**() uses just one object index for each data type and direction, so the size is limited to 252 byte in each direction. **EplApiProcessImageSetup**() can be easily enhanced to handle a larger process image.
If the application and the stack reside in the same address space, it is highly recommended to use **EplApiLinkObject**().

Note:

The different process variables are accessed in platform byte order. Thus on big endian machines mapping two adjacent UNSIGNED8 values to a PDO is not the same as mapping the same address as UNSIGNED16.

The following objects may be mapped to PDOs:

Object index	***Data Type***	***Direct-ion***	***Number of Sub indices***
0x2000	UNSIGNED8	input	EPL_API_PROCESS_IMAGE_SIZE_IN
0x2001	INTEGER8	input	EPL_API_PROCESS_IMAGE_SIZE_IN
0x2010	UNSIGNED16	input	EPL_API_PROCESS_IMAGE_SIZE_IN / 2
0x2011	INTEGER16	input	EPL_API_PROCESS_IMAGE_SIZE_IN / 2

Object index	***Data Type***	***Direct-ion***	***Number of Sub indices***
0x2020	UNSIGNED32	input	EPL_API_PROCESS_IMAGE_SIZE_IN / 4
0x2021	INTEGER32	input	EPL_API_PROCESS_IMAGE_SIZE_IN / 4
0x2030	UNSIGNED8	output	EPL_API_PROCESS_IMAGE_SIZE_OUT
0x2031	INTEGER8	output	EPL_API_PROCESS_IMAGE_SIZE_OUT
0x2040	UNSIGNED16	output	EPL_API_PROCESS_IMAGE_SIZE_OUT / 2
0x2041	INTEGER16	output	EPL_API_PROCESS_IMAGE_SIZE_OUT / 2
0x2050	UNSIGNED32	output	EPL_API_PROCESS_IMAGE_SIZE_OUT / 4
0x2051	INTEGER32	output	EPL_API_PROCESS_IMAGE_SIZE_OUT / 4

Table 29: Structure of static process image in OD

2.3.1.15 Function EplApiProcessImageExchangeIn()

Syntax:

```
#include <Epl.h>
tEplKernel PUBLIC EplApiProcessImageExchangeIn(
            tEplApiProcessImage*                pPI_p);
```

Parameters:

pPI_p:	Pointer to process image structure (see Table 30).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function replaces the passed input process image with the one of EPL stack.

Note:

In the implementation of EPL API Layer in Linux userspace (file EplApiLinuxUser.c) the function blocks until sync event for safely exchanging the process image. If the function returns the application is in the same state as if the sync callback function would be called. Thus the application must call EplApiProcessImageExchangeOut() as fast as possible to “return” from sync callback function.

```
typedef struct
{
    void*           m_pImage;
    unsigned int    m_uiSize;

} tEplApiProcessImage;
```

Member	***Description***
m_pImage	Pointer to process image
m_uiSize	Actual size of process image. It may be less than the one defined in EplCfg.h.

Table 30: Members of structure tEplApiProcessImage

2.3.1.16 Function EplApiProcessImageExchangeOut()

Syntax:

#include <Epl.h>

```
tEplKernel PUBLIC EplApiProcessImageExchangeOut(
            tEplApiProcessImage*                pPI_p);
```

Parameters:

pPI_p:	Pointer to process image structure (see Table 30).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function copies the passed output process image to EPL stack and marks TPDOs as valid.

Note:

In the implementation of EPL API Layer in Linux userspace (file EplApiLinuxUser.c) the function must be called only and in every case after calling EplApiProcessImageExchangeIn(). The function does not block.

2.3.2 Edrv module

The Edrv module is the Ethernet driver. Obviously it is target dependant.

2.3.2.1 Callback Function tEdrvRxHandler()

Syntax:

#include <edrv.h>

typedef void (*tEdrvRxHandler) (tEdrvRxBuffer * pRxBuffer_p);

Parameters:

pRxBuffer_p:	Pointer to receive buffer structure (see Table 31).

Return:

kEplSuccessful	The function was executed without error.

Description:

The callback function must be called by the Ethernet driver when Ethernet frames have been received. The DLL module provides this callback function. It is not reentrant.

```
typedef struct
{
    tEdrvBufferInFrame m_BufferInFrame;
    unsigned int       m_uiRxMsgLen;
    BYTE  *            m_pbBuffer;
    tEplNetTime        m_NetTime;

} tEdrvRxBuffer;
```

Member	*Description*
m_BufferInFrame	Position of received buffer in an Ethernet frame (see Table 32).
m_uiRxMsgLen	Length of received buffer in Bytes without the frame checksum.
m_pbBuffer	Pointer to the buffer, which is in any case the start of the Ethernet frame.

Member	***Description***
m_NetTime	Timestamp of Ethernet frame (see Table 3).

Table 31: Members of structure tEdrvRxBuffer

```
typedef enum
{
    kEdrvBufferFirstInFrame   = 0x01,
    kEdrvBufferMiddleInFrame  = 0x02,
    kEdrvBufferLastInFrame    = 0x04

} tEdrvBufferInFrame;
```

Value	***Description***
kEdrvBufferFirstInFrame	First data of frame received. Only used if EDRV_EARLY_RX_INT is defined as TRUE
kEdrvBufferMiddleInFrame	Middle data buffer of frame. Only used if EDRV_EARLY_RX_INT is defined as TRUE
kEdrvBufferLastInFrame	Last data of frame received, i.e. the complete frame was received. If EDRV_EARLY_RX_INT is defined as FALSE this value mus be used in any case.

Table 32: Constants of enumerated type tEdrvBufferInFrame

2.3.2.2 Callback Function tEdrvTxHandler()

Syntax:

#include <edrv.h>

typedef void (*tEdrvTxHandler) (tEdrvTxBuffer * pTxBuffer_p);

Parameters:

pTxBuffer_p:	Pointer to transmit buffer structure (see Table 33). This is the same pointer as previously used to start the transmission.

Return:

kEplSuccessful	The function was executed without error.

Description:

The callback function must be called by the Ethernet driver when an Ethernet frame has been transmitted. The DLL module provides this callback function. It is not reentrant. The macro EDRV_DMA_TX_HANDLER selects when this callback function is called exactly.

```
typedef struct
{
    unsigned int    m_uiTxMsgLen;
    unsigned int    m_uiBufferNumber;
    BYTE  *         m_pbBuffer;
    tEplNetTime     m_NetTime;
    unsigned int    m_uiMaxBufferLen;

} tEdrvTxBuffer;
```

Member	***Description***
m_uiTxMsgLen	Current length of the frame not including the frame checksum.
m_uiBufferNumber	Internal number of the buffer.
m_pbBuffer	Pointer to the buffer, i.e. the start of the Ethernet frame.
m_NetTime	Timestamp, when the frame was transmitted (see Table 3).
m_uiMaxBufferLen	Maximum length of buffer.

Table 33: Members of structure tEdrvTxBuffer

2.3.2.3 Function EdrvInit()

Syntax:

#include <edrv.h>

tEplKernel EdrvInit(
    tEdrvInitParam*   pEdrvInitParam_p);

Parameters:

pEdrvInitParam_p:	Pointer to initialization parameter structure

Return:

kEplSuccessful	The function was executed without error.

Description:

The function initializes the Ethernet driver and controller. The DLL module calls this function.

```
typedef struct
{
    BYTE             m_abMyMacAddr[6];
    tEdrvRxHandler   m_pfnRxHandler;
    tEdrvTxHandler   m_pfnTxHandler;

} tEdrvInitParam;
```

Member	***Description***
m_abMyMacAddr	Local MAC address.
m_pfnRxHandler	Rx handler (see section 2.3.2.1).
m_pfnTxHandler	Tx handler (see section 2.3.2.2).

Table 34: Members of structure tEdrvInitParam

2.3.2.4 Function EdrvShutdown()

Syntax:

#include <edrv.h>

tEplKernel EdrvShutdown(void);

Parameters:

none

Return:

kEplSuccessful	The function was executed without error.

Description:

The function shuts down the Ethernet driver and controller. The DLL module calls this function.

2.3.2.5 Function EdrvChangeFilter()

Syntax:

```
#include <edrv.h>
tEplKernel EdrvChangeFilter(
            tEdrvFilter *                   pFilter_p,
            unsigned int                    uiCount_p,
            unsigned int                    uiEntryChanged_p,
            unsigned int                    uiChangeFlags_p);
```

Parameters:

pFilter_p:	Pointer to array of filter structures.
uiCount_p:	Number of filter structures.
uiEntryChanged_p	Appoints a specific filter entry which has been changed. If it is greater than or equal to uiCount_p all entries have been changed.
uiChangeFlags_p	Specifies the type of changes which were made to the entry specified by uiEntryChanged_p. The following flags are valid: EDRV_FILTER_CHANGE_VALUE, EDRV_FILTER_CHANGE_MASK, EDRV_FILTER_CHANGE_STATE, EDRV_FILTER_CHANGE_ALL.

Return:

kEplSuccessful	The function was executed without error.
kEplEdrvTooManyFilters	uiCount_p is too large, i.e. the Edrv resp. the Ethernet Controller do not support this number of filter entries.

Description:

The function configures the specified Rx filter entries. Previously existing filters are overwritten. The Rx filter entries have to be ordered with the most specific filter entry at first. The first filter entry that matches to a received frame is used. The ordering is important particularly with regard to auto-response frames.

The Edrv may return a handle for each filter in the element m_uiHandle, that must be used for later calls to the function EdrvChangeFilter(). If the function is called with pFilter_p == NULL or uiCount_p == 0 existing Rx filters are removed.

The DLL module calls this function.

```
typedef struct
{
    unsigned int    m_uiHandle;
    BOOL            m_fEnable;
    BYTE            m_abFilterValue[22];
    BYTE            m_abFilterMask[22];
    tEdrvTxBuffer*  m_pTxBuffer;
#if (EDRV_FILTER_WITH_RX_HANDLER != FALSE)
    tEdrvRxHandler  m_pfnRxHandler;
#endif

} tEdrvFilter;
```

Member	***Description***
m_uiHandle	Handle of this filter entry which is returned by EdrvChangeFilter().
m_fEnable	Flag which specifies if this filter entry is enabled.
m_abFilterValue	Byte array of filter values.
m_abFilterMask	Byte array of filter mask.
m_pTxBuffer	Pointer to Tx buffer which will be transmitted by auto-response feature if the filter matches. If this pointer equals NULL, no Tx frame is triggered by this filter entry.
m_pfnRxHandler	Rx handler for this filter entry (see section 2.3.2.1).

Table 35: Members of structure tEdrvFilter

2.3.2.6 Function EdrvDefineRxMacAddrEntry()

Syntax:

#include <edrv.h>

tEplKernel EdrvDefineRxMacAddrEntry(
    BYTE*   pbMacAddr_p);

Parameters:

pbMacAddr_p:	Pointer to a multicast MAC address

Return:

kEplSuccessful	The function was executed without error.

Description:

The function is deprecated. It defines a multicast MAC address which shall be received. The DLL module calls this function.

2.3.2.7 Function EdrvUndefineRxMacAddrEntry()

Syntax:

```
#include <edrv.h>
tEplKernel EdrvUndefineRxMacAddrEntry(
            BYTE*                                   pbMacAddr_p);
```

Parameters:

pbMacAddr_p:	Pointer to a multicast MAC address

Return:

kEplSuccessful	The function was executed without error.

Description:

The function is deprecated. It undefines a multicast MAC address which must not be forwarded to the Rx handler anymore. The DLL module calls this function.

2.3.2.8 Function EdrvAllocTxMsgBuffer()

Syntax:

```
#include <edrv.h>
tEplKernel EdrvAllocTxMsgBuffer(
            tEdrvTxBuffer*                          pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function allocates a transmit buffer with the specified maximum size for the specified EPL frame type. The function returns a pointer to a buffer which is not less than the specified maximum size. The DLL module calls this function.

2.3.2.9 Function EdrvReleaseTxMsgBuffer()

Syntax:

```
#include <edrv.h>

tEplKernel EdrvReleaseTxMsgBuffer(
            tEdrvTxBuffer*                    pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function releases a previously allocated transmit buffer. The DLL module calls this function.

2.3.2.10 Function EdrvUpdateTxMsgBuffer()

Syntax:

```
#include <edrv.h>
tEplKernel EdrvUpdateTxMsgBuffer(
            tEdrvTxBuffer*                    pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function signals to the Edrv module, that the Tx frame was updated by the caller. The Tx frame has to be allocated previously via the function EdrvAllocTxMsgBuffer(). The DLL module calls this function.

2.3.2.11 Function EdrvSendTxMsg()

Syntax:

```
#include <edrv.h>

tEplKernel EdrvSendTxMsg(
            tEdrvTxBuffer*                    pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function transmits the specified buffer immediately. The caller must set the current size of the frame in m_uiTxMsgLen. Only previously allocated transmit buffers may be passed to this function. The DLL module calls this function.

2.3.2.12 Function EdrvTxMsgReady()

Syntax:

```
#include <edrv.h>

tEplKernel EdrvTxMsgReady(
            tEdrvTxBuffer*                    pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function marks the specified buffer as ready for transmission and start transferring the frame to the FIFO of the Ethernet controller but must not transmit it. The caller must set the current size of the frame in m_uiTxMsgLen. Only previously allocated transmit buffers may be passed to this function. This function may be present only if EDRV_FAST_TXFRAMES is defined as TRUE. The DLL module calls this function.

2.3.2.13 Function EdrvTxMsgStart()

Syntax:

```
#include <edrv.h>

tEplKernel EdrvTxMsgStart(
            tEdrvTxBuffer*                          pBuffer_p);
```

Parameters:

pBuffer_p:	Pointer to a transmit buffer structure (see Table 33).

Return:

kEplSuccessful	The function was executed without error.

Description:

The function really starts the transmission of the specified buffer which was previously marked as ready. This function may be present only if EDRV_FAST_TXFRAMES is defined as TRUE. The DLL module calls this function.

3 Object Dictionary

3.1 Fundamentals

The Object Dictionary (OD) forms the essential connection between the application software and the EPL stack, which enables data to be exchanged with an application over the EPL network. EPL defines the services and communication objects for the access to the OD entries. The OD of EPL is modeled after the one of CANopen. Each entry is addressed by index and sub index. The properties of an entry in the OD are defined by type (UINT8, UIN16, REAL32, Visible String, Domain,…) and by attribute (read-only, write-only, const, read-write, mappable).

The OD can contain up to 65536 index entries and 0 – 255 sub indexes per index. They are predefined by communication profile or device profile. Type and attribute for sub indexes within an index can vary.

Entries can be preset with default values. It is possible to modify the value of an entry with the help of SDOs (Service Data Objects), in as far as it is allowed by the attribute (read-write and write-only; not for read-only and const). A value can also be modified by the application itself (attribute read-write, write-only and read-only; not for const).

3.2 Structure of an OD, Standardized Profiles

The OD is divided into sections. The section 0x1000 – 0x1FFF is used to define the parameters for the communication objects and for storage of general information (manufacturer, device type, serial number, etc.). The entries from index 0x2000 – 0x5FFF are reserved for the storage of manufacturer specific entries. The entries starting at 0x6000 are those with device specific as described by the applicable device profile.

3.2.1 Communication Profile

The Ethernet POWERLINK V2.0 Communication Profile Specification defines the communication parameters for the communication objects, which must be supported by every EPL device. In addition, for device specific expansions to the communication profile applicable CANopen device profiles may be used.

3.2.2 Device Profiles

Overview of CANopen device profiles (not complete):

- Device profile for generic input/output modules (CiA 401)
- Device profile for drives and motion controls (CiA 402)
- Device profile for human/machine interface (HMI) (CiA 403)
- Device profile for measuring devices and closed-loop controllers (CiA 404)
- Device profile for encoders (CiA 406)
- Device profile for proportional valves (CiA 408)

For a complete overview of all currently available device profiles go to:
http://www.can-cia.org/downloads/ciaspecifications/

3.3 Object Dictionary Structure

The Object Dictionary consists of index tables, sub index tables, default values and the data itself.

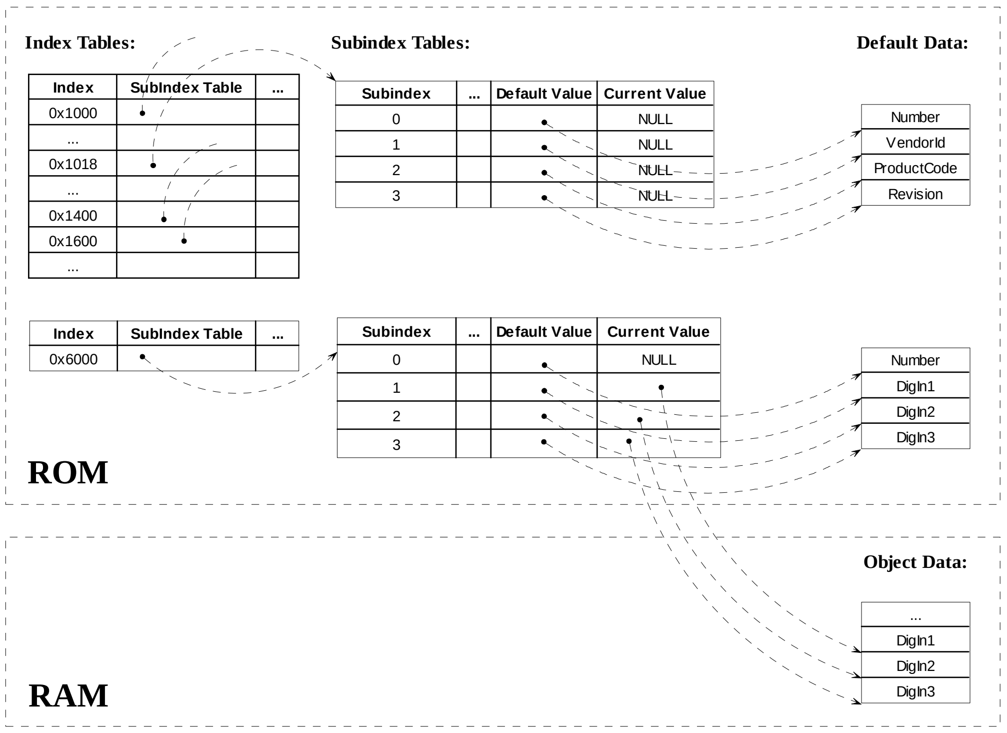

Figure 2: Object Dictionary Structure

The index tables contain all available indexes in ascending order. Each index entry refers to the beginning of the corresponding sub index table and its callback functions, which are called for each access.

Each sub index table contains all available sub indexes for an index entry. Every sub index entry provides information about the object type, the access rights, the location of the default value and the location of the actual data in RAM.

When the EPL stack is initialized, then the corresponding default values are assigned to all the object data.

The above described structures are constructed via macros. These macros are described in the following section.

3.4 Object Dictionary Definition

The Object Dictionary is defined in the file **objdict.h** by using macros, which are described in this section.

EPL_OBD_BEGIN()
EPL_OBD_END()

The Object Dictionary is always introduced with the macro EPL_OBD_BEGIN. EPL_OBD_END ends the definition of the Object Dictionary. No other macros can be used for the Object Dictionary outside of the boundary set by EPL_OBD_BEGIN and EPL_OBD_END.

EPL_OBD_BEGIN_PART_GENERIC()
EPL_OBD_BEGIN_PART_MANUFACTURER()
EPL_OBD_BEGIN_PART_DEVICE()
EPL_OBD_END_PART()
The macros EPL_OBD_BEGIN_PART_GENERIC, EPL_OBD_BEGIN_PART_DEVICE are always positioned between the macros EPL_OBD_BEGIN and EPL_OBD_END and introduce a partial section of the Object Dictionary. The "GENERIC" range is always utilized for the index range 0x1000 to 0x1FFF, the "MANUFACTURER" section is used for 0x2000 to 0x5FFF and the "DEVICE" section for index range 0x6000 to 0x9FFF. Each of these macros may only be used a single time within an Object Dictionary. The applicable range or partial section is always closed with the macro EPL_OBD_END_PART.

EPL_OBD_BEGIN_INDEX_RAM(ind,cnt,call)
EPL_OBD_END_INDEX(ind)
These macros are found within a partial section of the Object Dictionary. They are located within the range between the macros EPL_OBD_BEGIN_PART_… and EPL_OBD_END_PART. They must be ordered with ascending object index. These macros define an index entry in the Object Dictionary. An index entry is therefore always introduced with the macro EPL_OBD_BEGIN_INDEX_… and ended with EPL_OBD_END_INDEX. The suffix …_RAM indicates that the sub index table is located in RAM.

ind:	Object index of the entry to be defined
cnt:	Number of sub indexes within this index entry

call: Pointer to the callback function for this index entry

The callback function is always called if an object has been read or written. It doesn't matter if the access comes from the application or per SDO. The EPL API Layer has one callback function that must be specified for some objects in the index range 0x1000 through 0x1FFF and may be specified for any other object indexes, including application specific objects.

EPL_OBD_SUBINDEX_RAM_VAR(ind,sub,typ,acc,dtyp,name,val)
EPL_OBD_SUBINDEX_RAM_VAR_RG(ind,sub,typ,acc,dtyp,name,val,low,high)
EPL_OBD_SUBINDEX_RAM_VAR_NOINIT(ind,sub,typ,acc,dtyp,name)
EPL_OBD_SUBINDEX_RAM_VSTRING(ind,sub,acc,name,size, val)
EPL_OBD_SUBINDEX_RAM_OSTRING(ind,sub,acc,name,size, val)
EPL_OBD_SUBINDEX_RAM_DOMAIN(ind,sub,acc,name)
EPL_OBD_SUBINDEX_RAM_USERDEF(ind,sub,typ,acc,dtyp,name,val)
EPL_OBD_SUBINDEX_RAM_USERDEF_RG(ind,sub,typ,acc,dtyp,name,val,low, high)
EPL_OBD_SUBINDEX_RAM_USERDEF_NOINIT(ind,sub,typ,acc,dtyp,name)

The sub indexes are now defined within an index entry. They are always located within the range between the macros EPL_OBD_BEGIN_INDEX_… and EPL_OBD_END_INDEX and must be ordered with ascending sub index. Since there are various object types and therefore various data types that have to be created, there are different macros as well. The most important object types are:

- variables (VAR),
- strings (VSTRING for visible strings and OSTRING for octet strings),
- domains (DOMAIN) and
- user specific type (USERDEF).

The variables that were created with the macro ...RAM_VAR are objects that have a defined data length, which is determined by the object type (e.g. UNSIGNED 8, UNSIGNED 16, INTEGER 8, etc.). These objects cannot be mapped to a PDO.

Strings in RAM contain an additional parameter "size". This parameter indicates the maximum length of a string that can be written to via SDO or from the application.

The data length of domains is set by the application and can change during runtime.

Objects, which the user wants to manage in his application can be created with the USERDEF macros. Only these objects can be mapped to a PDO as process variables.

The suffix …_RG is for objects that have a value range for the object data. If in the file **EplCfg.h** the define EPL_OBD_CHECK_OBJECT_RANGE is set to TRUE, then the EPL stack automatically checks the value range before an object is written to (from the application or per SDO). If the define EPL_OBD_CHECK_OBJECT_RANGE is set to FALSE, then the value range is not checked automatically. Here the value range must be checked in the callback function of the object entry (in so far as this is necessary).

The suffix …_NOINIT defines objects which have no default value. That means that the EPL stack does not initialize those variables with a default value on NMT reset events. It is the responsibility of the application to initialize those objects.

The macros ..._DOMAIN, ..._USERDEF and ..._USERDEF_RG define in the RAM, along with the sub index entry, a variable information structure of type tEplVarEntry. This structure contains the data length and a pointer to the data. Upon initialization of the EPL stack with the function **EplApiInitialize()** all variable information gets deleted. The application has to map these objects to its own variables, by calling the function **EplApiLinkObject()**. All objects, which were created with the macros …_DOMAIN, …_USERDEF or …_USERDEF_RG must be defined via this function.

ind:	Object index of the sub index entry to be defined

sub:	Sub index for the sub index entry to be defined
typ:	Object type as code, as defined according to EPL V2.0 (*refer to Table 36*)
acc:	Access properties or access rights for the object (*refer to Table 37*)
dtyp:	Data type as C construction (*refer to Table 39*)
name:	Object name
size:	Maximum length of the string in RAM (incl. 0 termination)
val:	Default value for the object data, which is assumed following a reset
low:	Lower range limit for the object data
high:	Upper range limit for the object data

The parameters 'val', 'low' and 'high' must be given according to data type 'dtyp'.

EPL_OBD_RAM_INDEX_RAM_ARRAY(ind,cnt,cal,typ,acc,dtyp,name,def)
EPL_OBD_RAM_INDEX_RAM_VARARRAY(ind,cnt,cal,typ,acc,dtyp,nam e,def)

To simplify the definition of arrays there exist another class of macros. They replace the EPL_OBD_BEGIN_INDEX_…, EPL_OBD_END_INDEX and EPL_OBD_SUBINDEX_… macros.
These macros reduce the allocation of const memory, because of less sub index table entries. The drawback is that they need a little more RAM.

The macro …VARARRAY is the replacement of the EPL_OBD_SUBINDEX_RAM_USERDEF… macros. As such all created sub indexes must be defined by calling the function **EplApiLinkObject()**.

ind:	Object index of the sub index entry to be defined
cnt:	Number of sub indexes of this entry without sub index 0
cal:	Pointer to the callback function for this index entry
typ:	Object type as code, as defined according to EPL V2.0 (*refer to Table 36*)

acc:	Access properties or access rights for the object (*refer to Table 37*)
dtyp:	Data type as C construction (*refer to Table 39*)
name:	Object name
def:	Default value for the object data, which is assumed following a reset

Data Type in EPL	Meaning in EPL V2.0
kEplObdTypBool	Boolean (value 0x0001)
kEplObdTypInt8	signed integer 8-bit (value 0x0002)
kEplObdTypInt16	signed integer 16-bit (value 0x0003)
kEplObdTypInt32	signed integer 32-bit (value 0x0004)
kEplObdTypUInt8	unsigned integer 8-bit (value 0x0005)
kEplObdTypUInt16	unsigned integer 16-bit (value 0x0006)
kEplObdTypUInt32	unsigned integer 32-bit (value 0x0007)
kEplObdTypReal32	real 32-bit (value 0x0008)
kEplObdTypVString	visible string (value 0x0009)
kEplObdTypOString	octet string (value 0x000A)
kEplObdTypTimeOfDay	time of day (value 0x000C)
kEplObdTypTimeDiff	time difference (value 0x000D)
kEplObdTypDomain	domain (value 0x000F)
kEplObdTypInt24	signed integer 24-bit (value 0x0010)
kEplObdTypReal64	real 64-bit (value 0x0011)
kEplObdTypInt40	signed integer 40-bit (value 0x0012)
kEplObdTypInt48	signed integer 48-bit (value 0x0013)
kEplObdTypInt56	signed integer 56-bit (value 0x0014)
kEplObdTypInt64	signed integer 64-bit (value 0x0015)
kEplObdTypUInt24	unsigned integer 24-bit (value 0x0016)
kEplObdTypUInt40	unsigned integer 40-bit (value 0x0018)
kEplObdTypUInt48	unsigned integer 48-bit (value 0x0019)
kEplObdTypUInt56	unsigned integer 56-bit (value 0x001A)
kEplObdTypUInt64	unsigned integer 64-bit (value 0x001B)

Table 36: Available data types (enumerated type tEplObdType)

Access Rights	Value	Description

kEplObdAccRead	0x01	The object data can be read.
kEplObdAccWrite	0x02	The object data can be written to.
kEplObdAccConst	0x04	The object data is constant.
kEplObdAccPdo	0x08	The object data can be mapped to a PDO (always in conjunction with kEplObdAccVar).
kEplObdAccRange	0x20	The object data contains a value range.
kEplObdAccVar	0x40	The object contains a variable information structure.
kEplObdAccStore	0x80	The object data can be saved in EEPROM using the Store command.

Table 37: Access rights of objects

Combinations of access rights are possible (*refer to Table 38*). Some of these access rights automatically add the macros. Which objects contain which access rights depends on the applied device profile or on the application.

With Read/Write objects (kEplObdAccRead and kEplObdAccWrite set) there is always a value available in ROM, which contains the default value following Power On or Restore (object 0x1011) as well as a current value in RAM, which can be written and read for both SDO accesses or from the application.

Read-Only objects (macro EPL_OBD_SUBINDEX_RAM_… set but **kEplObdAccWrite** not set), however, can not be written per SDO. However, the application can modify its object data by calling the function **EplApiWriteLocalObject()**. For this there is a value created in ROM as well as in RAM.

Macro	**automatically assigned rights**
EPL_OBD_SUBINDEX_RAM_VAR	none
EPL_OBD_SUBINDEX_RAM_VAR_RG	kObdAccRange
EPL_OBD_SUBINDEX_RAM_VSTRING	none
EPL_OBD_SUBINDEX_RAM_OSTRING	none
EPL_OBD_SUBINDEX_RAM_DOMAIN	kObdAccVar
EPL_OBD_SUBINDEX_RAM_USERDEF	kObdAccVar

EPL_OBD_SUBINDEX_RAM_USERDEF_RG	kObdAccVar | kObdAccRange

Table 38: Automatically assigned access rights

Data Type in EPL	**Available Data Types as C Construct**
kEplObdTypBool	tEplObdBoolean
kEplObdTypInt8	tEplObdInteger8
kEplObdTypInt16	tEplObdInteger16
kEplObdTypInt24	tEplObdInteger24
kEplObdTypInt32	tEplObdInteger32
kEplObdTypInt40	tEplObdInteger40
kEplObdTypInt48	tEplObdInteger48
kEplObdTypInt56	tEplObdInteger56
kEplObdTypInt64	tEplObdInteger64
kEplObdTypUInt8	tEplObdUnsigned8, BYTE
kEplObdTypUInt16	tEplObdUnsigned16, WORD
kEplObdTypUInt24	tEplObdUnsigned24
kEplObdTypUInt32	tEplObdUnsigned32, DWORD
kEplObdTypUInt40	tEplObdUnsigned40
kEplObdTypUInt48	tEplObdUnsigned48
kEplObdTypUInt56	tEplObdUnsigned56
kEplObdTypUInt64	tEplObdUnsigned64
kEplObdTypReal32	tEplObdReal32
kEplObdTypReal64	tEplObdReal64
kEplObdTypTimeOfDay	tEplObdTimeOfDay
kEplObdTypTimeDiff	tEplObdTimeDifference
kEplObdTypVString	tEplObdVString
kEplObdTypOString	tEplObdOString
kEplObdTypDomain	all

Table 39: Available data types and their C counterparts

3.5 Example

The directory ObjDicts contains several sample object dictionaries. A good starting point might be ObjDicts/Api_CN.

4 Configuration and Scaling

The EPL stack is configured via C-defines in the header file ***EplCfg.h***. Because the configuration depends on the target and the current project, this file should reside in the project directory. The various configuration options are described in this section.

4.1 General configuration of the EPL stack

EPL_MODULE_INTEGRATION

The different modules of the EPL stack can be enabled or disabled separately. But some modules depend on each other. This macro is a bit field, where each bit represents a module. The following macros for each module exist:

EPL_MODULE_OBDK	= OBD kernel module
EPL_MODULE_PDOK	= PDO kernel module
EPL_MODULE_PDOU	= PDO user module
EPL_MODULE_SDOS	= SDO Server module
EPL_MODULE_SDOC	= SDO Client module
EPL_MODULE_SDO_ASND	= SDO via ASnd module
EPL_MODULE_SDO_UDP	= SDO via UDP module
EPL_MODULE_SDO_PDO	= SDO in PDO module
EPL_MODULE_NMT_MN	= NMT MN module (enabled MN support)
EPL_MODULE_NMT_CN	= NMT CN module
EPL_MODULE_NMTU	= NMT user module
EPL_MODULE_NMTK	= NMT kernel module
EPL_MODULE_DLLK	= DLL kernel module
EPL_MODULE_DLLU	= DLL user module
EPL_MODULE_OBDU	= OBD user module
EPL_MODULE_CFGMA	= Configuration Manager module
EPL_MODULE_VETH	= Virtual Ethernet driver module

EPL_USE_SHAREDBUFF

This macro defines if the SharedBuff implementation exists and should be used for event queues etc. If you want to test the EPL stack on a target system without operating system or on a target where no SharedBuff implementation exists, you can define this macro to FALSE. Instead of event queues there will be direct calls between EPL user part and kernel part.

EPL_EVENT_USE_KERNEL_QUEUE

This macro defines if the kernel part event queue is used. If this macro is defined to FALSE the kernel part event queue is replaced by direct calls within interrupt lock. This can provide a better performance, but increases the interrupt jitter.

4.2 Ethernet driver

EDRV_MAX_TX_BUFFERS

This macro defines the number of available transmit buffers in the Ethernet driver. A pure CN just needs 5 transmit buffers (i.e. its PRes frame, the IdentResponse, the StatusResponse, the NMT Request FIFO and non-EPL frames). A MN needs additionally 2 transmit buffers for the SoC and SoA frame and one transmit buffer for the PReq frame of each attached CN.

EDRV_AUTO_RESPONSE

This macro defines if the used Ethernet driver supports the auto-response feature. This means that the Ethernet controller will automatically transmit the response frames on certain requests, e.g. PRes on PReq and StatusResponse on StatusRequest.

EDRV_FAST_TXFRAMES

If the configured Ethernet driver supports it and this macro is defined as TRUE, fast transmit frames may be used for PRes frames. This means that the Ethernet driver implements the functions EdrvTxMsgReady() and EdrvTxMsgStart(). The first function starts copying the buffer to the Ethernet controller’s FIFO and second actually starts transmission of the message. This option may reduce the PReq-PRes latency. This macro depends on the implementation of the Ethernet driver.

EDRV_EARLY_RX_INT

If the configured Ethernet driver supports it and this macro is defined as TRUE, the Rx handler of the DLL module may be called before the complete frame has been received. This option may reduce the PReq-PRes latency because the DLL module can react on some received frames like PReq-frames if it only knows the header of the frame. This macro depends on the implementation of the Ethernet driver.

EDRV_USED_ETH_CTRL

This macro selects the Ethernet controller.

EDRV_DMA_TX_HANDLER

Selects if the Ethernet driver calls the Tx handler of the DLL module when the DMA transfer has finished or when the frame is actually transmitted over the bus. This macro depends on the implementation of the Ethernet driver.

4.3 DLL module

EPL_DLL_PRES_FILTER_COUNT

This macro specifies the number of PRes receive filter entries that the DLL shares with the Ethernet driver for PRes filtering. There are separate filter entries for PRes from different nodes. If the PDO module requests the reception of a PRes frame from a specific node, because the user has configured a RPDO with this node-ID, the DLL uses a free PRes filter entry for this PRes. For example if you have three RPDOs, you need at least 3 PRes receive filter entries. The source node-IDs of the specific PRes frames are configured at runtime.
A value of 0 disables the reception of cross-traffic, i.e. PRes frames from other nodes in the network are not seen by this node.

A value of -1 disables the selective PRes filtering. That means if the user enabled one or more RPDOs, so the PDO module requests the reception of the corresponding PRes frames, the DLL will enable the reception of all PRes frames. This implies that any PRes frame will be seen by the node. If there is no enabled RPDO, the PRes filter will be disabled.

EPL_DLL_PRES_READY_AFTER_SOC

If this macro and EDRV_FAST_TXFRAMES are defined as TRUE, the DLL module passes the PRes frame on SoC via EdrvTxMsgReady() to Ethernet driver. Currently it is only implemented in the CN state machine.

EPL_DLL_PRES_READY_AFTER_SOA

If this macro and EDRV_FAST_TXFRAMES are defined as TRUE, the DLL module passes the PRes frame on SoA via EdrvTxMsgReady() to Ethernet driver. Currently it is only implemented in the CN state machine.

The above macros are mutually exclusive.

4.4 OBD module

EPL_OBD_USE_KERNEL

If this macro is defined as TRUE, the OBD module implementation of EPL kernel part is also used in EPL user part via direct calls.

4.5 SDO modules

EPL_SDO_MAX_CONNECTION_ASND

This macro defines the maximum number of available connection channels of the SDO ASnd layer.

EPL_SDO_MAX_CONNECTION_UDP

This macro defines the maximum number of available connection channels of the SDO UDP layer.

EPL_MAX_SDO_SEQ_CON

This macro defines the maximum number of available connection channels of the SDO sequence layer.

EPL_MAX_SDO_COM_CON

This macro defines the maximum number of available connection channels of the SDO command layer.

4.6 Timer module

EPL_TIMER_USE_HIGHRES

If this macro is defined as TRUE, a high resolution timer module is available and will be used by DLL module for cycle monitoring.

4.7 EPL API Layer

EPL_API_PROCESS_IMAGE_SIZE_IN

The macro defines the size of static input process image. If it is set to 0 the static input process image is disabled. The value shall not exceed 252 and be a multiple of 4.

EPL_API_PROCESS_IMAGE_SIZE_OUT

The macro defines the size of static output process image. If it is set to 0 the static output process image is disabled. The value shall not exceed 252 and be a multiple of 4.

Glossary

AMI	Abstract memory interface
ASnd	EPL frame type: Asynchronous Send, which may contain SDO or NMT messages
CAL	Communication Abstraction Layer, internal EPL stack module
CN	Controlled Node, i.e. slave device in the EPL network
DCF	Device configuration file (generated by configuration tools)
DLL	Data Link Layer
DNS	Domain Name System (Internet Protocol)
EPL	Ethernet POWERLINK
EPSG	Ethernet POWERLINK Standardization Group
IP	Internet Protocol
HMI	Human machine interface
MAC	Media Access Control
MN	Managing Node, i.e. master device in the EPL network
NMT	Network Management
Node	an arbitrary EPL device. Often an EPL CN
OBD	Object dictionary module
OD	Object dictionary
PDO	Process Data Object
PReq	EPL frame type: Poll Request
PRes	EPL frame type: Poll Response

RPDO	Receive PDO
SDO	Service Data Object
SoA	EPL frame type: Start of Asynchronous
SoC	EPL frame type: Start of Cyclic
TCP	Transmission Control Protocol
TPDO	Transmit PDO
UDP	User Datagram Protocol

References

[1] EPSG Draft Standard 301 Ethernet POWERLINK Communication Profile Specification, Version 1.1.0, Ethernet POWERLINK Standardisation Group, 2008

[2] L-1108 Introduction into openPOWERLINK, SYS TEC electronic GmbH, Greiz, 2009

Document:	openPOWERLINK: Ethernet POWERLINK Protocol Stack
Document number:	L-1098e_3, Edition JuneSeptember 2009

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number:	
Name:	
Company:	
Address:	

Return to:

SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY
Fax : +49 (0) 36 61 / 62 79 99

Published by

SYS TEC ELECTRONIC

© SYS TEC electronic GmbH 2009

Ordering No. L-1098e_3
Printed in Germany